Opinnäytetyö (AMK)

Tietotekniikka

Ohjelmistotuotanto

2013

Jesse Toivonen

PHONEGAP-TYÖVÄLINE-OHJELMISTOJEN KEHITYS ÄLYPUHELIMIIN

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Ohjelmistotuotanto

10.06.2013 | 56

Ferm Tiina

Jesse Toivonen

PHONEGAP-TYÖVÄLINEOHJELMISTOJEN KEHITYS ÄLYPUHELIMIIN

Kilpailun vuoksi älypuhelimilla on erilaisia käyttöliittymillä ja eri ohjelmointikieliä. Tämä hankaloittaa kehittäjien arkea, kun pitäisi tuoda sovellus suuren käyttäjäjoukon ulottuville. Tätä ongelmaa varten on kehitetty erinäisiä tekniikoita, joilla voisi sovittaa sovellus useammalle käyttöjärjestelmälle ja laitteelle ilman, että pitäisi kirjoittaa ohjelmaa uudelleen uudelle kielelle. Tällainen lähestymistapa tietenkin tuo hyviä puolia ja huonoja puolia, joiden mukaan pitäisi sovellusta kehittää, mikäli lähtee käyttämään jotain näistä tekniikoista.

PhoneGap on yksi tekniikoista, joka tuo selaimien vahvuudet älypuhelinten sovellusten tekemiseen. Tämän tekniikan avulla sovellus voidaan kirjoittaa vain kertaalleen ja tuoda se sitten useaan älypuhelimelle, mutta tekniikassa on omat heikkoutensa selainpohjaisuuden takia. Selain on vain käyttöliittymärajapinta, jolloin kaikkia älypuhelinten sisäisiä toiminnallisuuksia ei saada käytettyä PhoneGapilla suoraan, vaan joudutaan kirjoittamaan se osa koodista älypuhelimen alkuperäisellä tekniikalla.

Tämän opinnäytetyön tavoitteena on tutkia, miten PhoneGap soveltuu älypuhelinten ohjelmien tekoon työvälineohjelmien tarpeita ajatellen. PhoneGapin tutkiminen tehtiin työvälinesovellusten tärkeimpien osa-alueiden osalta ostoslista-ohjelman avulla, jonka pääasiallisena tietovarastona toimii palvelimella oleva tietokanta. Vaikka ohjelma ei ole iso tietorakenteen osalta, siinä tulevat esille kaikki tärkeimmät työvälinesovelluksista osa-alueet, jotka ovat välttämättömiä mobiiliympäristössä, kuten synkronointi, internetyhteyden käyttö ja tiedon varastointi.

Opinnäytetyön tarkoituksena on myös esitellä kaikki PhoneGapin kirjastot, jotta sen mahdollisuudet tulevat paremmin esille. Tätä varten PhoneGapin tarjoamista kirjastoista tehtiin yhteen ohjelmaan niiden pääasialliset toiminnot. Kehitysalustana käytettiin Eclipseä, jolla tehtiin PhoneGapin avulla sovellukset Android-älypuhelinalustalle.

Tätä opinnäytetyötä voidaan käyttää referenssinä, kun ollaan päättämässä, käytetäänkö PhoneGapia älypuhelimen sovelluksen kehittämiseen vai tehdäänkö sovellus älypuhelimen omalla tekniikalla. PhoneGapilla voidaan tehdä tietynlaisia sovelluksia, mutta se ei voi hyödyntää aivan kaikkea älypuhelinalustojen tarjoamia mahdollisuuksia. Tämän takia pitäisi huomioida tekniikkaa valittaessa, mitkä ovat sovelluksen tarpeet.

ASIASANAT:

PhoneGap, HTML5, CSS3, javascript, älypuhelin, mobiili, sovelluskehitys

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Software Engineering

10 June 2013 | 56

Ferm Tiina

Jesse Toivonen

PHONEGAP–DEVELOPING TOOL SOFTWARE TO SMARTPHONES

Due to competition there are many varieties of smartphones, different operating systems and different programming languages. This makes programming troublesome and time consuming, when an application needs to be launched to large group of consumers. For solving this problem different kind of technologies have been developed, which can be used to port application to different smartphones without writing application to a new language. This kind of approach brings both pros and cons, which needs to be thought of before choosing this approach.

PhoneGap is one of these technologies and it brings internet browser's strengths to smartphone application development. Here code needs to be written once and bring it to multiple smartphones, but due to browser based development it has browser's weaknesses. Browser is only a user interface, in which case all smartphone's native features cannot be used with PhoneGap directly. If user wants to use smartphone's own native features, s/he needs to write that part of code to smartphone's own native form.

The target of this thesis was to study how PhoneGap fits in making of smartphone software in terms of tool software. In order to study PhoneGap in terms of tool software's most important sectors, a shopping list program was made. Here the main database runs in server. Although this program was small in terms of data structure, it has the important parts of tool software needed in mobile-environment, as synchronization, use of internet connection and storing of data.

However, as one function of this study is to present all PhoneGap's libraries to see its full potential, one program that includes all PhoneGap's offered libraries was made. Eclipse was used as development environment to make Android applications with PhoneGap.

This thesis can be used as reference when deciding of using PhoneGap or smartphone's own technique to develop application. With PhoneGap can be made certain kinds of applications, but it can't use all of smartphone platforms offered opportunities. Because of this you need to decide use of PhoneGap on what your application needs.

KEYWORDS:

PhoneGap, HTML5, CSS3, javascript, smartphone, mobile, software development

SISÄLTÖ

KUVAT

KÄYTETYT LYHENTEET

jQuery Mobile	jQuery Mobile on jQueryn tavoin rakennettu helpottamaan kehittäjien arkea. jQuery Mobile tuo kehittäjille helpon ja nopean mahdollisuuden toteuttaa nettisivujen ulkoasun ja toiminnallisuudet älypuhelinten käyttökokemusten tasolle. [1]
jQuery	jQuery on javascript-kirjasto, joka on kehitetty helpottamaan internetsivustojen kehittäjien ongelmaa eri internetselainten kanssa. Jokainen selain ei tue täydellisesti javascriptiä tai html:n virallisia kirjastoja. Aiemmin piti tehdä paljon erilaisia ns. purkkaratkaisuja, jotta sai jonkin asian toimimaan jokaisella selaimella. jQuery tuo siihen ratkaisun ja tuo kehittäjille kirjaston, jonka kautta pystyy kirjoittamaan samaa koodia kaikille selaimille. jQuery on testattu ja optimoitu. Tämä tuo kehittäjille rauhan sen suhteen, että kehittäjän pitäisi ottaa eri selaimet jokaisessa kohdassa huomioon. [2]
iOS	iOs on Applen kehittämä käyttöjärjestelmä älypuhelimille ja taulutietokoneille.
Android	Android on Android Inc.:n kehittämä käyttöjärjestelmä älypuhelimille ja taulutietokoneille, jota Google nykyään hallinnoi ostettuaan Android Inc:n.
JSON	JSON on tapa mallintaa tietoa merkkijonoina, jota käytetään yleensä palvelimen ja asiakasohjelman välisissä tiedonsiirroissa.
XML	XML on merkintäkieli, jolla tiedon kuvaus sisällytetään tietoon.
XAML	XAML on Microsoftin kehittämä XML johdannainen merkintäkieli, jolla voi kuvata tietorakenteita. Windows Phonen kehityksessä sillä kuvataan yleensä käyttöliittymässä olevia komponentteja.
HTML	HTML on merkintäkieli, jolla tehdään internetsivuja.
LINQ	LINQ on Microsoftin kehittämä komponentti, jolla voidaan lisätä tavallisen koodin sisään SQL-lauseita.
SQL	SQL on kyselykieli tietokannoille.
ORM	ORM on ohjelmointitekniikka, jolla voidaan tuoda esim. tietokantapohjaiset tietovarastot olio-ohjelmointi paradigmaan.
SSL	SSL on kryptausprotokolla, jolla voidaan luoda turvallisia keskusteluja internetin ylitse.

TSL	TSL on SSL:n korvaava kryptausprotokolla, jolla voidaan luoda turvallisia keskusteluja internetissä.
HTML5	HTML5 on HTML:n merkintäkielen 5:n version lyhenne, joka sisältää merkintäkielen uudistusten lisäksi myös javascripin uudistuksia, kuten tuoden enemmän mahdollisuuksia käsitellä mediaa internetsivuilla ilman kolmannen osapuolen tekniikoita.
CSS3	CSS3 on CSS-tyylityskielen uusin versio.

1 JOHDANTO

Älypuhelimet ovat tuoneet helpotusta nykypäivän kiireelliseen elämänkuvaan, tuomalla ennen vain pöytäkoneissa olevat ohjelmat käytettäväksi myös liikkuvan ihmisen arkeen. Kilpailun, yleisen innovaation ja erilaisuuden myötä on tullut eri laitevalmistajia, jotka ovat lähteneet tuottamaan älypuhelimia ja kehittämään asioita eteenpäin, joskus yhteisten ja joskus omien näkökantojen kanssa. Näin älypuhelimeen on tullut monimuotoisuutta. Tämän voidaan katsoa alkavaksi Applen iPhonen myötä. iPhone toi kosketusnäytöt, yksinkertaisen käyttöliittymän ja monimutkaiset ohjelmat lähemmäs käytettävyydeltään sopivammaksi pienille puhelimille.

Eri laitevalmistajilla on käytössä erilaisia alustoja omille älypuhelimilleen, joille voi tehdä omia ohjelmia. Laitevalmistajien erilaiset alustat hankaloittavat yhden ohjelman tekemistä useammille alustoille.

Ohjelmointikielet voivat olla samoja tai samankaltaisia, mutta itse alustojen ohjelmointikirjastot ovat erilaisia. Vaikka logiikka olisi ohjelmalla samanlainen kaikilla alustoilla, niin se kuitenkin pitäisi kääntää kieleltä toiselle alustojen välillä, jotta se saadaan useammalle alustalle asennettua. Erilaisten ohjelmointikielien ja ohjelmistokirjastojen vuoksi on ruvennut tulemaan erilaisia kolmannen osapuolen ratkaisuja kehittäjien avuksi, jotta kehittäjät saisivat saman ohjelman nopeammin ja vaivattomammin useammalle alustalle. Älypuhelinten kehityksessä yksi näistä ratkaisuista on PhoneGap.

Tämän tutkielman tarkoituksena on tehdä selvitys PhoneGap-kehityskirjaston tuomista mahdollisuuksista älypuhelinten sovellusten kehitykseen. Mitä sillä voi tehdä verrattuna natiivisti tehtyihin työvälinesovelluksiin, mitkä ovat sen haitat sekä edut?

Tässä työssä käsitellään PhoneGapia työvälinesovellusten tärkeimpien osa-alueiden osalta, kuten synkronointi, internetyhteyden käyttö ja tiedon varastointi. Tutkielman aikana tehtiin ostoslista-ohjelma, jonka pääasiallisena tietovarasto-

na toimii palvelimella oleva tietokanta. Vaikka ohjelma sinällään ei ollut iso tietorakenteen osalta, siinä tulee kaikki tärkeimmät osa-alueet työvälinesovelluksista esille, joita tarvitsee olla mobiiliympäristössä. Tässä tutkielmassa tarkoituksena on myös käydä koko PhoneGapin ominaisuudet lävitse, jotta sen mahdollisuudet tulevat paremmin esille. Tämän takia tehtiin kaikista PhoneGapin tarjoamista kirjastoista toiminnot yhteen ohjelmaan.

2 ÄLYPUHELINALUSTAT

Älypuhelinalustoja on useita. Suosituimmat ovat iOs, Android ja Windows Phone. Tässä osiossa käsitellään vain nämä kolme alustaa, sillä ne ovat Suomessa kuin maailmallakin suosituimmat ja näin niillä on suurimmat käyttäjämäärät yhteensä.

2.1 iOs

iOs on Applen kehittämä älypuhelinten ja tablettien alusta, joka tunnetaan iPhonen ja iPadin alustana. iOs on vain käytössä Applen omissa tuotteissa ja näitä tuotteita ovat vain noin vuosittain päivittyvät iPhone ja iPad. iOs:n kanssa ei tarvitse ajatella ohjelmointia laitteiston kannalta oikein lainkaan, koska Apple on pienellä laitekannallaan poistanut sen tarpeen. iOs:n kirjastoilla tehdyt ohjelmat toimivat täten melkein joka laitteessa. [3].

2.2 Android

Android on älypuhelinten ja tablettien alustana tunnettu käyttöjärjestelmä. Androidin on kehittänyt loppupäässä Google, mutta Google osti tämän alustan pienemmältä yritykseltä 2005. Android pohjautuu Linux-ytimeen, ja Google on antanut laitevalmistajille vapauden muokata alustaa tiettyyn pisteeseen asti. Tästä johtuen Android-alustalle sovelluksia kehitettäessä, pitää olla tarkkana mille kaikille laitteille haluaa oman sovelluksensa julkaistavan, jotta kaikki toiminnot toimivat halutusti. [4]

2.3 Windows Phone 7

Windows Phone on Microsoftin kehittämä käyttöjärjestelmä älypuhelimille. Sitä voivat älypuhelinvalmistajat käyttää omien älypuhelintensa käyttöjärjestelmänä.

Tällä hetkellä HTC:llä, Samsungilla ja Nokialla on Windows Phone 8-puhelimia markkinoilla.[5]

3 TYÖVÄLINEOHJELMISTOJEN TARPEET

Työvälineohjelmistoilla tarkoitetaan yleensä sellaisia sovelluksia, joita käytetään päivittäisessä työssä. Kirjoitusohjelmat, laskutusohjelmat, kaavio-ohjelmat jne. eli yleishyödylliset ohjelmat ovat työvälineohjelmia. Älypuhelimien työvälineohjelmilla on yleensä perustarpeet, joita ne tarvitsevat toimiakseen kaikkialla. Sillä älypuhelin on suunniteltu liikkuvalle ihmiselle ja näin ollen myös ohjelmien suunnittelussa pitää muistaa, että käyttäjä ei aina ole samassa paikassa. Tällöin tarvitaan jotain, jolla pitää käyttäjä ajan tasalla. [6]

3.1 Tiedon säilytys

Käsiteltävää tietoa pitää pystyä säilyttämään myös silloin, kun käyttäjä ei käytä ohjelmaa eli aja sitä, jolloin tieto säilyy käyttömuistissa. Käyttäjän lopettaessa ohjelman käytön tiedon pitää tallentua johonkin, sillä käyttömuistissa oleva tieto menetetään ohjelman sammuessa. Älypuhelimissa tähän soveltuu kaksi tiedon säilytystapaa, internetissä oleva palvelin tai säilytetään tieto puhelimen muistissa. Internetissä oleva palvelin on hyvä tiedon säilytys, jos tarvitsee käyttää jaettua sisältöä, jota päivitetään muualtakin kuin vain puhelimesta. Puhelimen muistiin voi alustasta riippuen tallentaa moneenkin tapaan ja moneen formaattiin. Yleisin ja kätevin tapa on tietokanta, johon voidaan tallentaa tietoa suuria määriä ja samalla pitää se järjestyksessä, josta voidaan hakea nopeasti haluttu tieto. [7]

Tiedon säilytyksessä pitää muistaa, ettei säilyttäisi puhelimessa mitään arkaa tietoa, kuten sosiaaliturvatunnuksia tai muita tietoja, jotka voivat edesauttaa varasta päästäen hänet esim. yrityksen verkkoon tai hän voisi tehdä jopa identiteettivarkauden. Tällöin pitää ajatella, mitä voidaan siirtää puhelimelle ja miten se säilytetään siellä: onko se puhdasta tietoa vai kryptattua tietoa. Kryptaus on tiedon salausta. Kryptauksessa tehdään ymmärrettävästä tiedosta sekavaa,

jonka kuitenkin voi esim. salausavaimella purkaa taas uudestaan ymmärrettävään muotoon. [8]

3.2 Internetyhteys

Jos kehitetään ohjelmistoa, jonka pitää päästä yleiseen tietovarastoon, joka on jollakin palvelimella, siihen tarvitaan internetyhteyttä. Muuten ei pystytä pitämään puhelimen tietoja ja palvelimen tietoja ajan tasalla.

3.3 Synkronointi

Synkronointi on kahden eri paikassa olevan tiedon pitämistä ajan tasalla eli samanlaisina. Älypuhelimessahan ei internetyhteys ole aina päällä. Käyttäjä voi olla kuuluvuusalueen ulkopuolella tai sellaisella alueella, missä kuuluu vain gsm-verkko ja näin ollen ei pääse internetiin, joten on kätevää pitää osaa käsiteltävää tietoa puhelimen muistissa, jolloin sitä voi myös käyttää, vaikka yhteys katkeaisi. Näin ollen synkronointi on hyvin oleellinen asia älypuhelinten työvälinesovelluksissa.

Siirrettävän tiedon määrä pitää muistaa pitää pienenä, koska käytetään älypuhelimia, jotta käyttäjän ei tarvitse odottaa pitkiä aikoja, kun siirretään tietoja palvelimelle tai palvelimelta puhelimelle. Tämän takia pitää tehdä valinta käytettävän tiedonsiirtoformaattien ja protokollien välillä. [9]

4 ÄLYPUHELINALUSTOJEN KOMPONENTTIEN EROT

Tässä luvussa käsitellään edellä mainittujen työvälinohjelmistoissa tärkeimmiksi havaittujen toimintojen erot Androidin, iOs:n ja Windows Phone 7:n välillä.

4.1 iOs

4.1.1 Kehitysalustat

iOs:lle tehdään sovelluksia Applen kehittämän Xcode-kehitysalustan avulla. Yleensä joutuu asentamaan iOs-kehityspalikan itse, jonka jälkeen vasta pääsee tekemään iOs:llä sovelluksia. Kielenä käytetään myös Applen kehittämää Objective-C-kieltä, joka poikkeaa syntaksiltaan huomattavasti muihin kieliin verrattuna, vaikkakin yhtäläisyyksiä löytyy myös paljon. [10]

4.1.2 Tiedon säilytys

iOs:ssä pystyy säilyttämään tietoa kolmella eri tavalla. Lokaalisti tietoa pystyy tallentamaan kahdella tavalla omiin tiedostoihinsa suljetussa tietovarastossa tai SQLite-tietokantaan. Uutena tallennusmuotona on iCloud-pilvipalveluun tallentaminen. [11]

Lokaali tietokanta

iOs:ssä käytetään lokaalina tietokantana SQLite-tietokantaa, johon otetaan yhteys iOs:n CoreData:n kautta, joka on Applen tekemä komponentti tiedon käsittelyä varten tietokantojen kanssa. Toisin kuin Windows Phonessa ja Androidissa, kehittäjän ei tarvitse tehdä paljon omaa koodia tietokannan kanssa kommunikointiin, vaan voi suoraan käyttää tätä komponenttia. Eikä kehittäjän tarvitse kirjoittaa tietokannan mallinnusta käsin, vaan siihen käytetään Xcodesta löytyvää mallinnustyökalua. Näin kehittäjä säästää rutkasti aikaa. [12]

Lokaali suljettu tietovarasto

Kehittäjä voi myös tallentaa tietoa lokaalisti suljettuun vain sovelluksen omassa käytössä olevaan tietovarastoon. Tähänkin Apple on kehittänyt apukomponentteja, joilla lukeminen ja kirjoittaminen hoituvat käden käänteessä ja vieläpä XML-tiedostoihin. Tosin mikäli haluaa kirjoittaa tiedostoon, niin ensin pitää viedä tämä tiedosto kansioon, jossa on kirjoitusoikeudet, ennen kuin on sallittu kirjoittaa niihin. Koska ohjelman asennuksen jälkeen, kaikki tiedostot ovat yhdessä sovelluskansiossa, jonne ei voi kirjoittaa. [13]

iCloud

iCloud on Applen kehittämä pilvipalvelu, johon voi tuoda dokumentteja ja tietoa, mutta ei vielä käyttää ohjelmia. [11]

4.1.3 Internetyhteys ja tiedonvälitys

Tiedonvälitys hoituu iOs:ssä http-protokollan kautta tai salatun https-protokollan kautta, käyttäen GET- ja POST-metodeja. Välitykseen yleensä käytetään JSON-muodossa olevaa tietoa, jotta internetin käyttö olisi nopeampaa ja akkua säästävämpää. Kiitos Applelle iOs 5:een on tullut oma JSON-jäsentäjästä. Tästä johtuen ei tarvitse enää käyttää muiden tekemiä kirjastoja tähän. [14] iCloudia käyttäen synkronointi palvelimen päässä olevan tietokannan kanssa ei vielä onnistu, niin kuin Windows Azurella onnistuu tällä hetkellä. Eli tällainen synkronointi pitää hoitaa iOs:llä vielä alusta loppuun itse. Kirjoittaen kaiken palvelinpään palvelut tätä varten ja iOs:n pään rajapinnan.

4.2 Android

4.2.1 Kehitysalustat

Androidille tehdään ohjelmia pääasiassa käyttäen Eclipseä, johon on ladattu Androidkehitykseen tehty lisämoduuli, joka mahdollistaa Androidille kehittämisen Eclipsessä. [4] Kielenä Androidille sovelluksia kehittäessä käytetään Javaa, joka on Sun Microsystemsin kehittämä ohjelmointikieli, joka toimii pääasiassa

virtuaalikoneen avulla. Ensin kieli kirjoitetaan yhteen kertaan ja käännetään toiselle kielelle Java bytecodeksi, jota virtuaalikone osaa ajaa laitteistoriippumattomasti. Eli laitteistolla ei ole väliä, kunhan virtuaalikoneen saa asennettu sille. [15]

4.2.2 Tiedon säilytys

Androidillakin voidaan säilyttää koneen käyttämää tietoa moneen paikkaan ja monella tavalla. Lokaalisti ja omaan palvelimeen internetin välityksellä.

Lokaali tietokanta

Androidissa pystyy säilyttämään tietoa SQLite-tietokannassa aivan kuin iOs:kin. Tosin toteutus on erilainen. Ei ole mitään aputyövälinettä datamallin tekoon ja kehittäjä joutuu itse kirjoittamaan haut samantapaisella logiikalla kuin SQL:ssä, tosin ei kirjoiteta suoraa SQL:ää vaan käytetään olioita. Nämä käytettävät oliot omalta osaltaan helpottavat vähän kehittäjän työtä. [7]

Lokaali suljettu tietovarasto

Aivan kuin iOs:llä ja Windows Phone 7:llä myös Androidissa voidaan käyttää tiedon säilytykseen vain sovelluksen omassa käytössä olevaa tietovarastoa. Toteutus tähän on aika helppoa ja aika samantapaista kuin kehittäisi tavallista pc-ohjelmaa, jossa käsitellään tiedostoja ja kansioita. Tämän vuoksi kehittäjän on helppo mukautua tähän menetelmään, mikäli on kehittänyt ennenkin tällaisia tiedostotason ohjelmia. [7]

Ulkoinen tietovarasto

Androidissa voidaan myös tallentaa tietoa ulkoisessa tietovarastossa, kuten muistikortilla. Tässäkin toteutus on helppoa aivan kuin suljetun tietovaraston kanssa, koska lähestymistapa ei poikkea siitä. Vaan lähdetään taas ajattelemaan tiedostoja ja kansioita, menettämättä Javassa olevia tiedostoajattelutapaa ja käytetään samankaltaisia kirjastoja tiedon varastoinnissa kuin käytetään muutenkin Javassa. Tämä onkin helppo oppia juuri tämän ajattelutavan takia, mikäli on Javan kanssa ennen ollut tekemisissä. [7]

Palvelimella sijaitseva tietovarasto

Tässä Android poikkeaa muista siinä mielessä, että vielä ei ainakaan Androidilla ole omaa pilvipalvelua, vaan kehittäjä joutuu tekemään palvelinpuolen ympäristön tällä kertaa kokonaan aivan itse. Toteutus hoidetaan siten, että tehdään internetpalvelu johonkin palvelimeen, jolta sitten ohjelma voi kysyä tietoa tai lähettää tietoa. [7]

4.2.3 Internetyhteys ja tiedonvälitys

Androidillakin käytetään http- tai https-protokollia tiedonvälitykseen laitteen ja palvelimen välillä. Androidilta löytyy JSON-jäsennin valmiina, joten ei tarvitse kirjoittaa itse tai käyttää ulkopuolisten tekemiä jäsentimiä. Koska Androidilla ei ole omaa pilvipalvelua, niin toteutus on vähän vaivalloisempaa verrattuna muihin alustoihin.

4.3 Windows Phone 7

4.3.1 Kehitysalustat

Windows Phone 7:lle tehdään sovelluksia C# .NET- tai Visual Basic .Net-kielellä käyttäen hyödyksi .NET Frameworkkia ja Silverlight- ja/tai XNA-kehitysalustaa. [8]

Silverlight on Microsoftin kehittämä alun perin Flashin tapaiseen käyttöön tarkoitettu internetsivuille upotettava alusta, joka mahdollistaa rikkaiden internetsovellusten teon. Mutta Windows Phone 7:n myötä se irrotettiin osaksi Windows Phone 7:ää. Silverlightissa käyttöliittymää tehdään XAML-merkkikielellä, joka muistuttaa html:ää tai xml:ää osaksi. Vähän niin kuin Androidilla ja iOs:lla käytetään myös käyttöliittymään tekoon merkkikieltä. [16]

XNA on toinen Microsoft Phone 7:n kehitysalusta, joka on tarkoitettu pääasiassa pelien tekoon ja muuhun graafisiin toimintoihin, joihin ei Silverlightilla pysty, kuten 3D- ja vaativaan 2D -grafiikkaan. [17]

4.3.2 Tiedon säilytys

Windows Phone 7:llä on kolme tiedonsäilytystapaa kaksi lokaalisti toimivaa ja yksi, joka toimii Microsoftin Azureus pilvipalvelun kautta.

Lokaali tietokanta

Windows Phone 7 sai tuen lokaalille tietokannalle vasta Windows Phone OS 7.1 -version myötä 30.9.2011. Käytetään LINQ to SQL:ää ORM:nä, eli mallinnuskielenä, kun tehdään sovellusta. Windows Phone 7:ssä käytetään Androidin ja iOs:n tavoin eristettyä varastoa tallennetuille tiedoille, jota vain sovellus, jolle tämä hiekkalaatikko on tehty voi käyttää, eli käytetään ns. sovelluskohtaista tietovarastoa tietokannan säilyttämiseen. Tällä hetkellä ei vielä ole suunnitteluvälinettä datamallin tekoon LINQ to SQL muotoon, vaan pitää käsin kirjoittaa tietokannan rakenne ja toiminnot. Eli kehitystyö on jokseenkin hitaampaa verrattuna iOs:ään, jossa on jo suunnitteluväline tällaista tarkoitusta varten. [18]

Lokaali suljettu tietovarasto

Tietoa voidaan tallentaa Windows Phone 7:än lokaaliin suljettuun tietovarastoon, josta sitä voidaan hakea. Suosittu tallennusmuoto on XML-tiedostot, joihin voidaan tallentaa tietoa elementtien sisään näin helpottaen sen löytämistä. [18]

Azureus-pilvipalvelu

Azureus on Microsoftin kehittämä pilvipalvelualusta, jossa voidaan pyörittää ohjelmia ja myös tallentaa tietoja pilvessä olevaan tietokantaan, jolloin kaikki pilvessä olevat ohjelmat ja tieto on saatavilla kaikkialla ja näin ollen ei tarvitse synkronoida erikseen laitteen tietovaraston ja palvelimen tietovaraston välillä. [19]

4.3.3 Internetyhteys ja tiedonvälitys

Windows Phone 7:ssä tiedonvälitys internetin ylitse palvelimelle tapahtuu http-protokollan avulla POST- ja GET-metodeita käyttäen. Salattuun liikenteeseen käytetään https-protokollaa, joko SSL:ää tai TLS-ää käyttäen. [20] Mikäli tekee itse palvelinpään toiminnot, niin silloin kannattaa lähettää tiedot JSON-muodossa, joka on hyvin vähää tilaa vaativa ja näin ollen säästää akkua ja aikaa lähetyksessä sekä vastaanotossa, sillä silloin ei pidä pitää niin kauan internetyhteyttä auki. Mikäli käyttää Azureusta tällaisista palvelimen ja mobiililaitteen välisistä kommunikaatioista ei tarvitse huolehtia niin paljon, vaan silloin on syytä käyttää Windows Azure Toolkit for Windows Phone. Jolloin käytetään valmiiksi tehtyjä komponentteja ja voidaan vain kirjoittaa palvelinpään ohjelmia, jotka antavat tietoa tai ottavat sitä vastaan. [21]

5 PHONEGAP

PhoneGap on ohjelmistokehys älypuhelinten ohjelmien kehittämiseen, joka on luotu helpottamaan kehittäjien arkea ja mahdollistamaan yhden ohjelman tuominen useampaan älypuhelinalustaan pienemmällä vaivalla.

PhoneGapilla pystytään tekemään HTML5:lla, Javascriptilla ja CSS3:lla ohjelmia älypuhelimille, jotka toimivat yhdellä kirjoituskerralla kaikilla sen tukemilla älypuhelinkäyttöjärjestelmillä. [22]

PhoneGap tarjoaa jopa pilvipalveluna kääntäjäpalvelua, jotta ei tarvitsisi itse omata jokaiselle älypuhelinkäyttöjärjestelmälle omaa kehitysympäristöä, kääntäjää ja näin vielä omaa tietokonetta eritoten Windows Phone ja iOs ohjelmien kehittämiseen [23].

Siihen tässä tutkielmassa ei tutustuta, vaan tärkeintä on saada yleiskatsaus PhoneGappiin ja sen tuomiin mahdollisuuksiin ajatellen työvälineohjelmistoja.

5.1 Phonegapin asennus

Tässä luvussa käydään lävitse PhoneGapin asennus, kun kehitysympäristö on asennettuna koneelle, joka sisältää älypuhelimelle tarkoitetun SDK:n ja työkalut.

Esimerkeissä käytetään Androidia. Muille alustoille löytyy ohjeet suoraan tästä osoitteesta.

http://docs.phonegap.com/en/2.4.0/guide_getting-started_index.md.html#Getting%20Started%20Guides

Kaikilla alustoilla lähtölinjat ovat samat, kun kehitysympäristöt on asennettu. Ensiksi haetaan internetistä oikeat versiot kehityskirjastoista koneelle ja ne sisällytetään projektiin, jonka jälkeen ohjelman tekeminen voikin alkaa.

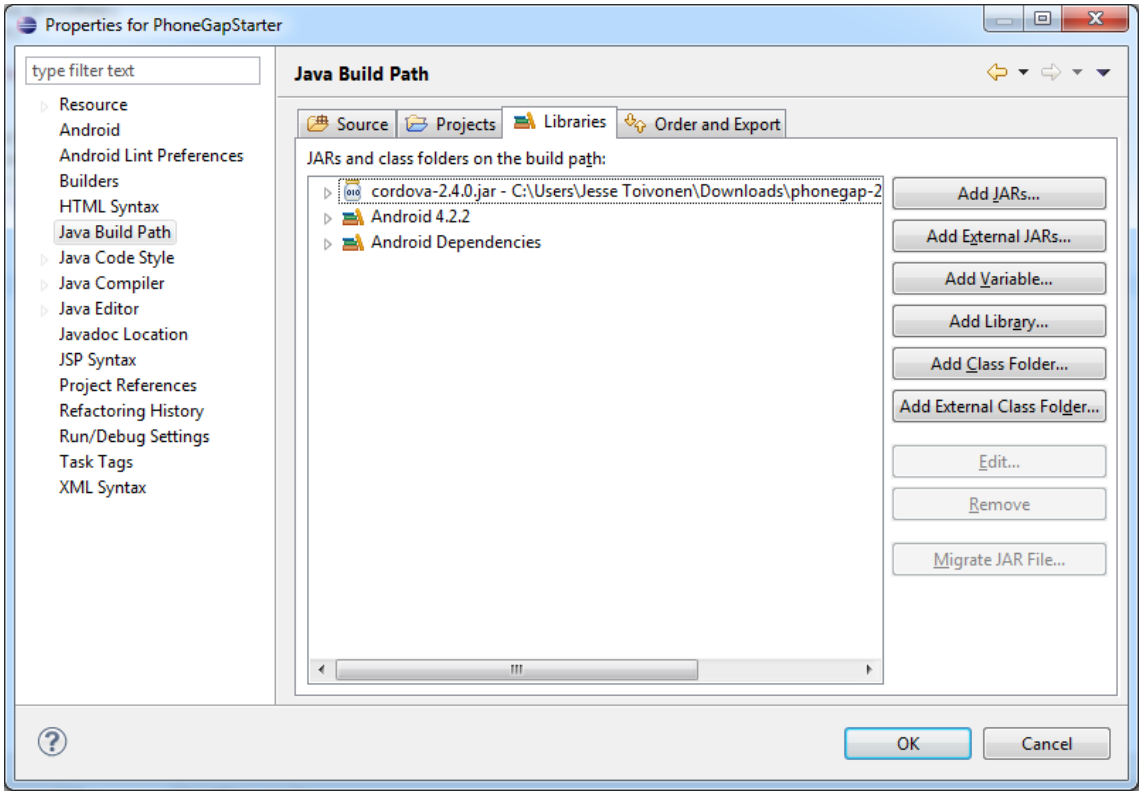
Tarvitset Phonegapista 2.4 version osoitteesta.

http://phonegap.com/download/

Kun PhoneGap on ladattu koneelle, puretaan paketti ja seurataan tiedostopolkua oikean älypuhelinkäyttöjärjestelmän kansioon esim. Android ..\phonegap-2.4.0\lib\android, josta löytyy tuolle käyttöjärjestelmälle tarkoitettu PhoneGap javascript-kirjasto sekä rajapintana toimiva kirjasto. Windows Phonelle WBCordovaClassLib.dll, Androidille cordova-2.4.0.jar ja iOs:lle on vähän erilainen, sillä sieltä löytyy oma hakemisto, joka sisällytetään PhongeGap projektiin, eikä pelkästään kahta tiedostoa.

Android kansion alta löydetyt tiedostot kopioidaan PhoneGap-projektiin cordova-2.4.0.js www-kansion js-alikansioon ja cordova-2.4.0.jar projektin lib-kansioon.

Tämän jälkeen pitää cordova-2.4.0.jar tiedosto sisällyttää projektin kirjastoihin ja sen saa tehtyä projektin asetuksista Project->Properties->Java Build Paths->Libraries alta. Kuvassa 1 on esimerkki jar-tiedoston lisäämisestä projektiin Eclipsellä.



Kuva 1. Kuva jar-paketin lisäämisestä projektiin.

Nyt tarvitsee ladata jQuery projektiin. Tämän hetken Phonegap 2.4 versioon tarvitset jQuerysta 1.8.2.

http://blog.jquery.com/2012/09/20/jquery-1-8-2-released/

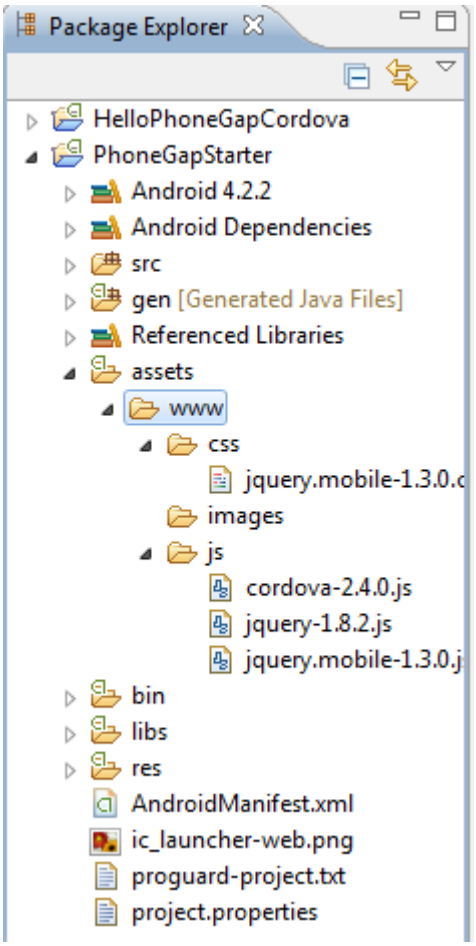
Kun jQuery on ladattu, kopioidaan se PhoneGap projektin www-kansion js-alikansioon.

Nyt tarvitsee ladata jQuery Mobile projektiin. Tämän hetken Phonegap 2.4 versioon tarvitset jQuery Mobilesta 1.2 version.

http://jquerymobile.com/download/

Kun jQuery Mobile on ladattu, huomataankin, että siellä ei olekaan pelkästään javascript tiedostoa vaan myös css-tyyli tiedosto. Javascript tiedosto kopioidaan www-kansion js-alikansioon ja css-tyyli tiedosto kopioidaan www-kansion css-alikansioon.

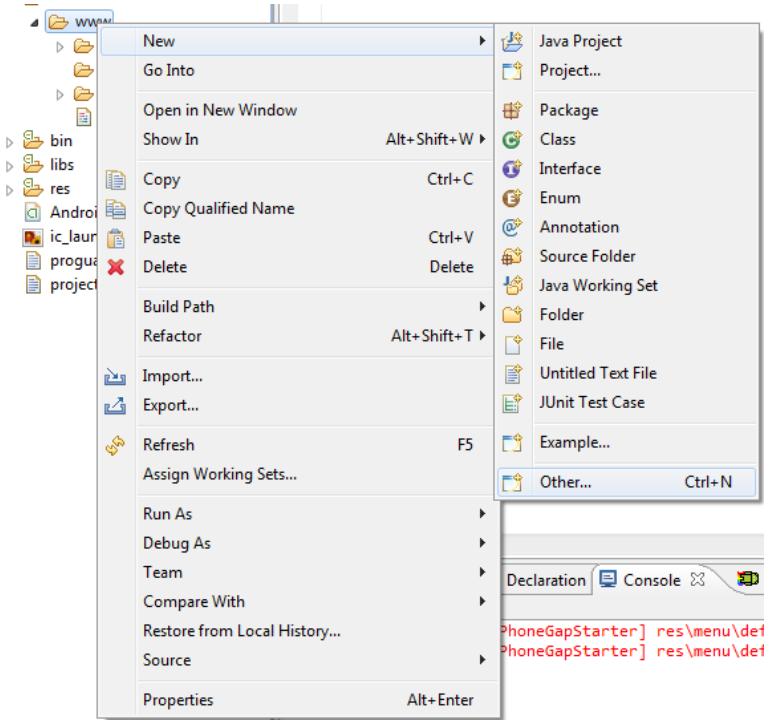
Kuvassa 2 on esimerkki hakemistorakenteesta, kun kaikki tarpeelliset tiedostot on sisällytetty projektiin.



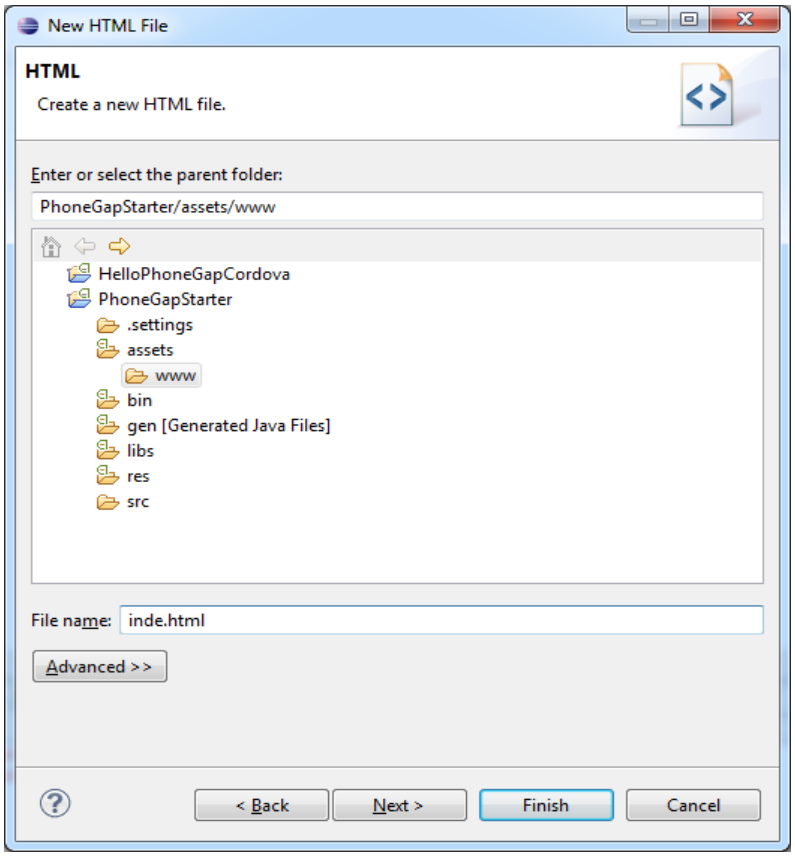
Kuva 2. Kuva hakemistorakenteesta, jossa on tarvittavat kirjastot sisällytetty.

Tämän kaiken latailun jälkeen, kun jokainen tiedosto löytyy Android projektista ja jos niitä ei heti näy siellä kansioiden sisällä Eclipsessä voidaan päivittää Eclipse projekti File->Refresh tai F5-pikanäppäimen avulla ja nyt pitäisi nuo kopioidut tiedosto jo näkyä. Nyt lisätään www-kansioon ohjelman etusivu index.html, jonka päälle aletaan kasata ohjelmaa. Sivun voi lisätä kansiota oikean hiiren korvan valikosta valitsemalla New->Other->Other->HTML File.

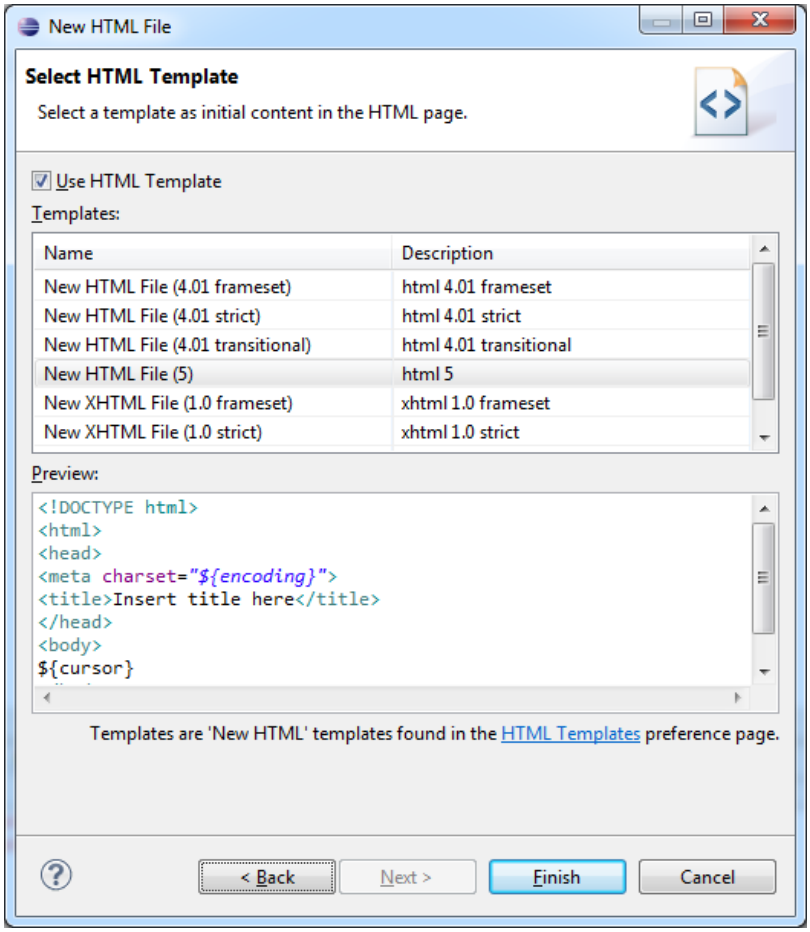
Kuvassa 3 on polku HTML-sivun lisäämisestä projektiin. Kuvassa 4 annetaan HTML-sivulle nimi. Kuvassa 5 valitaan HTML-sivun HTML:n versio, jonka mukaan internetselain osaa käyttää valitun HTML-version määrityksiä, miten sivu näytetään.



Kuva 3. Kuva uuden html-sivun lisäämisestä projektiin.



Kuva 4. Kuva uuden html-sivun lisäämisestä projektiin.



Kuva 5. Kuva uuden html-sivun lisäämisestä projektiin.

Kuvassa 6 etusivuun Index.html:ään sisällytetään nuo ladatut tiedostot lisäämällä niille linkkielementit, jotta selain tietää hakea ne tuolle sivulle.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>PhoneGapStarter</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script type="text/javascript" charset="utf-8" src="js/cordova-2.4.0.js"></script>
    <link rel="stylesheet" href="js/jquery.mobile-1.3.0.css" type="text/css" />
    <script type="text/javascript" charset="utf-8" src="js/jquery-1.8.2.js"></script>
    <script type="text/javascript" charset="utf-8" src="js/jquery.mobile-1.3.0.js"></script>
</head>
<body>

</body>
</html>
```

Kuva 6. Kuva ladattujen kirjastojen sisällyttämisestä html-sivulle.

Tämän kaiken jälkeen PhoneGap on valmiina käytettäväksi.

5.2 PhoneGapin kirjastot

PhoneGap sisältää monia eri kirjastoja, jokainen kirjasto on älypuhelimen tiettyihin toimintoihin tarkoitettu. Tässä osiossa käydään lävitse nuo kirjastot ja näytetään esimerkit muutamista, mitä käytin tekemässäni projektissa, jonka kautta tutustuin PhoneGappiin.

PhoneGapin kirjastoissa on hyvin tunnusomaista se, että ne on tehty ajatellen yksinkertaisuutta. Samaan tapaan kuin HTML5:n uudet javascript-kirjastot. Yksinkertaisuus perustuu näissä kirjastoissa pieneen määrään funktioita, joilla hoituu koko ominaisuuden käskyttäminen puhelimessa ilman, että joutuu ensin alustamaan oliota suurella määrällä tietoa ja sen jälkeen vasta pystytään käyttämään sitä.

Kirjastot käydään lävitse yksitellen esitellen niiden toiminnot yksinkertaisuudessaan, lisää tietoa kirjastoista saa aina PhoneGapin dokumentaatioista, jotka löytyvät erikseen jokaiselle PhoneGapin versiosta. Olen pyrkinyt tuomaan mahdollisimman yksinkertaisen kuvan siitä mitä näillä kirjastoilla voi tehdä, enkä ole käynyt liian yksityiskohtaisesti lävitse niitä, sillä niiden tuet muuttuvat aina eri älypuhelinkäyttöjärjestelmien myötä ja ne pidetään paremmin ajan tasalla PhoneGapin dokumentaatiossa. Jokaisen kirjaston esittelyn lopusta löytyy linkki kirjaston täydelliseen dokumentaatioon tässä dokumentissa käsiteltyyn 2.4 versioon.

Perusperiaate näillä kirjastoilla on se, että jokaisesta löytyy vain vähän funktioita, joilla voi hoitaa tarvittavat toiminnot. Melkein jokaiselle funktiolle annetaan parametreina funktio, joka käsittelee onnistuneen funktiokutsun, funktio, joka käsittelee epäonnistuneen funktiokutsun ja funktion optionaliset asetukset, joilla voidaan muokata funktion toimintaa.

Jokaisella eri kirjastolla on omat oikeusmääritteet, jotka tarvitsee asettaa sovelluksessa, jotta niitä voidaan käyttää. Tämä johtuu älypuhelinten käyttöjärjestelmistä, joissa on pyritty tekemään ohjelmille tiukat säännöt, mitä ne voivat tehdä. Näiden sallittujen toimintojen käyttö taas edellyttää oikeuksien asettamisen,

jonka käyttäjä hyväksyy ohjelmaa asentaessaan, jotta hän näkee sen mitä ohjelma tekee. Ottaa yhteyden verkkoon, käyttää puhelimen kontakteja jne..

5.2.1 Accelometer

Accelometer-kirjasto hoitaa puhelimen asennon kuuntelun x, y ja z akselilla maata kohden eli missä asennossa puhelin on maata kohden. Tällä kirjastolla voidaan kysyä puhelimelta sen nykyistä sijaintia tai kuunnella asennon muutoksi tietyn aikavälin välein. iOs ei tue asennon hakua tietyssä kohtaa vaan iOs:ssä joutuu aina kuuntelemaan puhelimen asentoa ja pitämään kirjaa sen asennosta.

Kuvassa 7 on kuuntelu esimerkki Accelometer-kirjaston käytöstä. Ensin odotetaan, että PhoneGap on latautunut, jonka jälkeen voidaan vasta aloittaa laitteen asennon kuuntelu.

```
var accelerationWatchID = null;

// Odotetaan, että PhoneGap latautuu
//
document.addEventListener("deviceready", onDeviceReady, false);

// PhoneGap on valmis
//
function onDeviceReady() {
    // Aloitetaan kuuntelu
    StartAccelerationWatch();
}

// Aloitetaan laitteen asentotiedon kuuntelu
function StartAccelerationWatch() {
    var options = { frequency: 3000 };
    accelerationWathcID = navigator.accelerometer.watchAcceleration(onAccelerationSuccess, onAccelerationError, options)
}

// Lopetetaan laitteen asentotiedon kuuntelu
function StopAccelerationWatch() {
    if (accelerationWatchID) {
        navigator.accelerometer.clearWatch(accelerationWatchID);
        accelerationWatchID = null;
    }
}
```

Kuva 7. Esimerkki Accelometer-kirjaston käytöstä, kun kuunnellaan laitteen asentotietoja.

Kuvassa 8 on esimerkki miten voidaan hyödyntää laitteen asennon kuuntelun tai haun tuomaa tulosta. Siinä näytetään käyttäjälle kaikki kuuntelun tai haun palauttaman tiedot.

```
// Kun laitteen nykyisten asentotietojen luku onnistuu, käsitellään ne.
function onAccelerationSuccess(acceleration) {
    $('#accelometerDiv').append('X-akseli: ' + acceleration.x + '<br />' +
          'Y-akseli: ' + acceleration.y + '<br />' +
          'Z-akseli: ' + acceleration.z + '<br />' +
          'Aika: '      + acceleration.timestamp/1000/60/60 + ':' + acceleration.timestamp/1000/60 + ':' + acceleration.timestamp/1000 + <br />');
}

// Kun laitteen nykyisten asentotietojen luku epäonnistuu, näytetään käyttäjälle PhoneGap ilmoitus.
function onAccelerationError() {
    navigator.notification.alert('Virhe Acceleration haussa!', null, "Oh noes!");
}
```

Kuva 8. Esimerkki Accelometer-kirjaston käytöstä, kun käsitellään onnistunut kuuntelu.

Tässä on linkki Accelometer-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_accelerometer_accelerometer.md.html

5.2.2 Camera

Camera-kirjasto hoitaa puhelimen kamera ohjelman käskyttämisen oman applikaation sisällä, joka mahdollistaa kuvan tai videon hakemisen joko hakemistoista tai kamerasta. Kun kuva tai video on otettu jostain, palataan takaisin omaan applikaation, jossa voidaan jatkaa kuvan käsittelyä. Camera-kirjasto sisältää kuvan tai videon ottamisen kamerasta, kuvakirjastosta tai kuva-albumista, kameran asetukset kuvan tai videon ottamishetkellä ja iOs:llä myös tilapäishakemistoon tallennettujen kuvien ja/tai videoiden poiston.

Tiedoston hakeminen hoituu yhdellä metodilla camera.getPicture(), joka hoitaa kuvan tai videon hakemisen sille annetuista asetuksista riippuen.

Väliaikaisten tiedostojen tyhjentäminen iOs:llä hoituu myös yhdellä metodilla camera.cleanup.

Hakemisen aikana voidaan seuraavat asetukset asettaa kameralle:

- **quality:** Kuvanlaatu 0 – 100 asteikolla
- **destinationType:** Funktiokutsun palautus tyyppi base64 enkoodattu tai tiedoston tallennuspolku
- **sourceType:** Kuvan lähde, josta kuva otetaan, joko kuvakirjasto, kamera tai kuva-albumi
- **allowEdit:** Sallitaan kuvan yksinkertainen editointi ennen kuin kuva otetaan

- **encodintType:** Kuvan palauttamis enkoodaus, jpeg tai png-muodossa
- **targetWidth:** Kuvan leveys, johon kuva skaalataan, pitää käyttää targetHeight-asetuksen kanssa
- **targetHeight:** Kuvan korkeus, johon kuva skaalataan, pitää käyttää target-Width-asetuksen kanssa
- **mediaType:** Määrää sen minkälaisesta mediasta valitaa. Toimii vain, kun on lähdetyypiksi valittuna kuvakirjasto tai kuva-albumi.
- **correctOrentation:** Kääntää kuvan laitteen asemaan nähden oikeaksi.
- **saveToPhotoAlbum:** Kuvan tai videon ottamisen jälkeen määrää sen tallennetaanko kuva tai video albumiin.
- **popoverOption:** Vain iOs:llä toimivat asetukset, jotka määräävät popoverin kohdan albumissa tai kirjastossa.

Kuvassa 9 on esimerkki Camera-kirjaston käytöstä. Kun esimerkki funktio suoritetaan, aukaistaan natiivisovellus kuvien hakuun, josta käyttäjä valitsee kuvan. Kun kuva on valittu, näytetään se kuva käyttäjälle.

```
// Kuvan hakemiseen tarkoitettu funktio, jota voidaan käyttää esim. painikkeen kanssa
function GetPicture() {
    // Haetaan kuva kuva-albumista, jolloin laitteen natiiviohjelma antaa valita
    // kuvan albumista ja me saamme tiedon siitä mitä tapahtui, tässä tapauksessa
    // kuvan tiedostopolun.
    navigator.camera.getPicture(onSuccess, onFail, { quality: 50,
        destinationType: Camera.DestinationType.FILE_URI,
        sourceType: pictureSource.SAVEDPHOTOALBUM });
}

// Mikäli kuvan hakeminen onnistui näytetään kuva.
function onSuccess(imageURI) {
    var image = document.getElementById('imgPhotoAlbum');
    image.src = imageURI;
}

// Mikäli kuvan hakeminen ei onnistunut näytetään käyttäjälle PhoneGap ilmoitus.
function onFail(message) {
    navigator.notification.alert('Kuvan hakeminen epäonnistui koska: ' + message, null, "Oh noes!");
}
```

Kuva 9. Esimerkki Camera-kirjaston käytöstä kuvan hakemiseen kuva-albumista.

Tässä on linkki Camera-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_camera_camera.md.html#Camera

5.2.3 Capture

Capture-kirjastolla hoidetaan Camera-kirjastosta poiketen vain kuvan, äänen tai videon ottaminen laitteella avaamalla laitteen alkuperäisen kamera-ohjelman, josta saadaan palautteena tietoa otetuista kuvista, videosta tai äänestä. Kuvan, äänen ja videon ottamiseen on omat metodinsa ja niillä on omat asetuksensa.

Kuva

Kuvassa 10 on esimerkki Capture-kirjaston käytöstä, kun halutaan ottaa kuva. Kuvan ottamiseen annetaan myös valinnainen asetus limit, joka rajoittaa kuvien ottamisen yhteen kuvaan. Kun kuva on otettu, käydään kaikki kuvat lävitse, sillä captureImage palauttaa aina joukon kuvia, vaikka rajoitin olisikin yhdessä kuvassa. Virheen sattuessa näytetään käyttäjälle ilmoitus.

```
//Aukaistaan natiivisovellus kuvan ottamiseen
function CaptureImageWithDevice() {
    navigator.device.capture.captureImage(onCaptureImageSuccess, onCaptureImageError, {limit: 1});
}

// Mikäli saatiin otettua kuva, näytetään käyttäjälle kuvatiedoston tiedot.
function onCaptureImageSuccess(mediaFiles) {
    var i, len;
    for (i = 0, len = mediaFiles.length; i < len; i += 1) {
        showImageFiles(mediaFiles[i]);
    }
}

// Mikäli tulee virhe kuvan ottamisessa näytetään PhoneGapin ilmoitus.
function onCaptureImageError(error) {
    navigator.notification.alert('Tapahtui virhe kuvan ottamisessa.', null, "Oh noes!");
}
```

Kuva 10. Esimerkki Capture-kirjaston käytöstä kuvan ottamiseen.

Video

Kuvassa 11 on esimerkki Capture-kirjaston käytöstä videon nauhoittamiseen. Esimerkissä annetaan valinnainen asetus limit, joka rajoittaa videon ottamisen yhteen videoon. Videon nauhoittamisen jälkeen käydään kaikki tallennetut tiedostot lävitse ja näytetään ne käyttäjälle. Virheen sattuessa näytetään käyttäjälle ilmoitus.

```
// Aukaistaan natiivisovellus videon tallentamiseen
function CaptureVideoWithDevice() {
    navigator.device.capture.captureVideo(onCaptureVideoSuccess, onCaptureVideoError, {limit: 1})
}

// Mikäli saatiin nauhoitettua video, näytetään käyttäjälle videotiedoston tiedot.
function onCaptureVideoSuccess(mediaFiles) {
    var i, len;
    for (i = 0, len = mediaFiles.length; i < len; i += 1) {
        showVideoFiles(mediaFiles[i]);
    }
}

// Mikäli tulee virhe videon nauhoittamisessa näytetään PhoneGapin ilmoitus.
function onCaptureVideoError(error) {
    navigator.notification.alert('Tapahtui virhe videon nauhoittamisessa.', null, "Oh noes!");
}
```

Kuva 11. Esimerkki Capture-kirjaston käytöstä videon ottamiseen.

Ääni

Kuvassa 12 on esimerkki Capture-kirjaston käytöstä äänen nauhoittamiseen. Esimerkissä annetaan valinnainen asetus limit, joka rajoittaa äänen nauhoittamisen kahteen tiedostoon. Nauhoittamisen jälkeen äänitiedostot näytetään käyttäjälle. Virheen sattuessa näytetään käyttäjälle ilmoitus.

```
// Aukaistaan natiivisovellus äänen äänettämiseen.
function CaptureAudioWithDevice() {
    navigator.device.capture.captureAudio(onCaptureAudioSuccess, onCaptureAudioError, {limit: 2});
}

// Mikäli saatiin äänitetty ääntä, näytetään käyttäjälle äänitiedoston tiedot.
function onCaptureAudioSuccess(mediaFiles) {
    var i, len;
    for (i = 0, len = mediaFiles.length; i < len; i += 1) {
        showAudioFile(mediaFiles[i]);
    }
}

// Mikäli tulee virhe äänen äänettämisessä näytetään PhoneGapin ilmoitus.
function onCaptureAudioError(error) {
    navigator.notification.alert('Tapahtui virhe äänen äänittämisessä.', null, "Oh noes!");
}
```

Kuva 12. Esimerkki Capture-kirjaston käytöstä äänen ottamiseen.

Tässä on linkki Capture-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_media_capture_capture.md.html#Capture

5.2.4 Compass

Compass-kirjastolla saadaan suunta, johon laite osoittaa palautettuna asteina (0-359.99). Compassilla voidaan joko ottaa tämän hetkinen suunta tai kuunnella suunnan vaihtoja tietyllä annetulla aikavälillä. Compass-kirjaston kuuntelija ja nykypaikan hakevat metodit molemmat palauttavat compassHeading olion, josta voidaan hakea kompassin suunta (heading) asteina sekä kulma (bearing) asteina, johon siis kompassin nuoli osoittaa.

Kuvassa 13 on kaksi esimerkkiä Compass-kirjaston käytöstä. Ensimmäisessä esimerkissä WatchCompassHeading kuunnellaan kompassin muutoksia kolmen sekunnin välein, jonka jälkeen tulos näytetään käyttäjälle. Toisessa esimerkissä GetCompassHeading-funktiossa haetaan kompassin suunta vain kerran. Virheen sattuessa näytetään käyttäjälle ilmoitus.

```
var compassWatchID = null;

// Kuunnellaan kompassin suuntaa
function WatchCompassHeading() {
    var options = { frequency: 3000 };
    compassWatchID = navigator.compass.watchHeading(onCompassSuccess, onCompassError, options);
}

// Pysäytetään kompassin suunnan kuuntelu.
function StopCompassHeadingWatch() {
    if (compassWatchID) {
        navigator.compass.clearWatch(compassWatchID);
        compassWatchID = null;
    }
}

// Haetaan kompassin suunta.
function GetCompassHeading() {
    navigator.compass.getCurrentHeading(onCompassSuccess, onCompassError);
}

// Mikäli kompassin suunnan haku onnistuu, näytetään se käyttäjälle.
function onCompassSuccess(heading) {
    $('#divHeading').append('Suunta asteissa: ' + heading.magneticHeading);
}

// Mikäli tulee virhe kompassin suunnan haussa, näytetään PhoneGap ilmoitus.
function onCompassError(error) {
    navigator.notification.alert('Kompassin suunnan haussa tapahtui virhe.', null, "Oh noes!");
}
```

Kuva 13. Esimerkki Compass-kirjaston käytöstä.

Tässä on linkki Compass-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_compass_compass.md.html#Compass

5.2.5 Connection

Connection-kirjastolla saadaan tietoa laitteen yhteyksien tilasta. Puhelinyhteyksien ja langattoman internetyhteyksien tilasta.

Kuvassa 14 on esimerkki Connection-kirjaston käytöstä. Esimerkissä haetaan älypuhelimen verkkoyhteyden tila. Tiloja on seitsemän erilaista.

```
// Tarkastetaan verkon tila ja näytetään se käyttäjälle.
function checkConnection() {
    var verkonTila = navigator.connection.type;

    var tilat = {};
    tilat[Connection.UKNOWN] = 'Tuntematon yhteys';
    tilat[Connection.ETHERNET] = 'Langallinen yhteys';
    tilat[Connection.WIFI] = 'Langaton yhteys';
    tilat[Connection.CELL_2G] = '2G yhteys';
    tilat[Connection.CELL_3G] = '3G yhteys';
    tilat[Connection.CELL_4G] = '4G yhteys';
    tilat[Connection.NONE] = 'Ei yhteyttä';

    $('#divTila').html(tilat[verkonTila]);
}
```

Kuva 14. Esimerkki Connection-kirjaston käytöstä.

Tässä on linkki Connection-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_connection_connection.md.html#Connection

5.2.6 Contacts

Contacts-kirjastolla päästään käsiksi laitteen kontakteihin. Niitä voidaan hakea ja luoda. Kontakteja käsitellään luonnissa ja haussa Contact-olion kautta, joka ilmentää kontaktia, sisältäen globaalin tunnisteen, nimen, puhelinnumerot, sähköpostiosoitteet jne. eli kaiken mitä puhelimessa voi näkyä kontaktilla, kun selailee yhteystietoja.

Haku

Kontaktien haku hoidetaan contacts.find(contactFields, contactSuccess, contactError, contactFindOptions)-funktiolla. Funktion contactFields-parametrina annetaan taulukkona ne kentät kontaktista, jotka palautetaan Contact-oliossa. ContactSuccess-parametrina annetaan sen funktion nimi, joka käsittelee onnistuneen funktiokutsun. ContactError-parametrina annetaan sen funktion nimi, joka käsittelee epäonnistuneen funktiokutsun ja contactFindOptions-parametrina annetaan ContactFindOptions oliona, johon voidaan asettaa haun ehdot, filtteri ja tosi/epätosi arvo haetaanko useampia kontakteja.

Kuvassa 15 on esimerkki kontaktien hausta ja niiden käsittelystä.

Luonti

Kontaktin luominen hoidetaan contacts.create(properties)-funktiolla, joka palauttaa uuden Contact-olion, jolle voidaan asettaa sitten kontaktin tiedot, joko suoraan properties-parametrilla tai olion attribuutteihin asettamalla. Contacts.create-funktio ei luo kontaktia suoraan puhelimen kontakteihin vaan sitä varten pitää Contact-oliosta kutsua save-funktiota, joka tallentaa uuden kontaktin puhelimen kontakteihin.

Kuvassa 15 on esimerkki kontaktin luomisesta ja sen attribuuttien asetuksesta.

Contact-luokka

Contact-luokka sisältää seuraavat attribuutit, jotka määrittävät kontaktin:

- **id**: Klobaali tunniste kontaktille.
- **displayName**: Kontaktin nimi, jota voidaan näyttää käyttäjälle
- **name**: Olio ContactName, joka sisältää kontaktin täydellisen nimen, sukunimen, toisen nimen, etunimen, nimen liitteen mr tai ms, nimen lisän Von..
- **nickname:** Kasuaali nimi kontaktille, jota voidaan käyttää
- **phoneNumbers:** Taulukko ContactField-olioista, joka sisältää puhelinnumerot, kuten työ-, koti- tai mobiilinumeron ja niiden välillä kontaktin suositellun numeron.
- **emails:** Taulukko ContactField-olioista, joka sisältää sisältää kontaktin sähköpostiosoitteet tyypeittäin, kuten koti ja työ-sähköpostiosoitteet.

- **addresses:** Taulukko ContactAddress-olioista, joka sisältää tiedot kontaktin osoitteista tyypeittäin, kuten koti ja työ-osoitteet.
- **ims:** Taulukko ContactField-olioista, joka sisältää kontaktin IM osoitteet tyypeittäin.
- **organizations:** Taulukko ContactOrganization-olioista, jotka sisältävät kontaktin organisaatiot yritykset yms..
- **birthday:** Kontaktin syntymäpäivä Date-oliona.
- **note:** Kontaktin lisätiedot.
- **photos:** Kontaktin kuvien tiedot ContactField-olioina.
- **categories:** Taulukko ContactField-olioista, jotka sisältävät käyttäjän määrittelemät kategoriat kontaktille.
- **urls:** Taulukko ContactField-olioista, jotka sisältävät kontaktiin liittyvät internet-sivustot.

```
// Luo kontaktin ja näyttää sille lisätyt nimen ja puhelinnumeron käyttäjälle.
function LuoKontakti() {
    var kontakti = navigator.contacts.create({"displayName": "Testi Käyttäjä"});
    kontakti.phoneNumbers[0] = new ContactField('mobile', '0402121321', true);
    $('#divContact').html('Kontaktin nimi: ' + kontakti.displayName + '<br />'
            + 'Kontaktin puhelin: ' + kontakti.phoneNumbers[0].value);
}

// Etsitään kontakia
function EtsiKontakti() {
    var options = new ContactFindOptions();
    options.filter = 'Jesse';
    options.multiple = false;
    navigator.contacts.find(["*"], onContactSuccess, onContactError, options);
}

// Mikäli kontaktien haku onnistui, näytetään käyttäjälle kontaktien nimet.
function onContactSuccess(contacts) {
    for(var i = 0; i < contacts.length; i++) {
        $('#divContact').append('Nimi: ' + contacts[i].displayName);
    }
}

// Mikäli tapahtui virhe kontaktien haussa näytetään käyttäjälle PhoneGap ilmoitus.
function onContactError(error) {
    navigator.notification.alert('Tapahtui virhe kontaktien haussa.', null, 'Oh noes!');
}
```

Kuva 15. Esimerkki Contacts-kirjaston käytöstä.

Tässä on linkki Contacts-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_contacts_contacts.md.html#Contacts

5.2.7 Device

Device-kirjastolla päästään käsiksi laitteen tietoihin, kuten laitteen nimeen, PhoneGapin versio, käyttöjärjestelmä, universaali tunniste laitteelle, käyttöjärjestelmän versio ja laitteen mallin nimen.

Kuvassa 16 on esimerkki Device-kirjaston käytöstä ja siitä mitä eri tietoa Device-kirjaston kautta voi löytää.

```
// Näytetään käyttäjälle laitetiedot
function NaytaLaitetiedot() {
    $('#divDevice').html('Nimi: ' + device.name + '<br />'
            + 'PhoneGapin versio: ' + device.cordova + '<br />'
            + 'Käyttöjärjestelmä: ' + device.platform + '<br />'
            + 'ID: ' + device.uuid + '<br />'
            + 'Versio: ' + device.version + '<br />'
            + 'Malli: ' + device.model + '<br />');
}
```

Kuva 16. Esimerkki Device-kirjaston käytöstä.

Tässä on linkki Device-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_device_device.md.html#Device

5.2.8 Events

Events-kirjasto sisältää PhoneGapin tapahtumat, joihin voidaan asentaa kuuntelija ja näin tehdä jotain, kun tapahtuma laukeaa. Tapahtumiin voi liittää kuuntelijoita, kun PhoneGap on kokonaan latautunut. Ensin pitää siis odottaa, että deviceready-tapahtuma laukeaa, jonka jälkeen voidaan vasta muihin tapahtumiin asettaa kuuntelijat.

Kuvassa 17 on esimerkki Events-kirjaston käytöstä. Esimerkissä asetetaan deviceready tapahtumaan kuuntelija, jonka sisältämät komennot suoritetaan tapahtuman lauettua.

Tapahtumat

- **deviceready:** laukeaa, kun PhoneGap on kokonaan ladattu
- **pause:** laukeaa, kun applikaatio menee taustalle

- **resume:** laukeaa, kun applikaatio tulee taustalta aktiivikseksi
- **online:** laukeaa, kun internetyhteys aukeaa
- **offline:** laukeaa, kun internetyhteys menee kiinne
- **backbutton:** laukeaa, kun käyttäjä painaa laitteen paluu-näppäintä
- **batterycritical:** laukeaa, kun PhoneGap havaitsee, että akun lataustaso on saavuttanut ns. kriittisen pisteen
- **batterylow:** laukeaa, kun PhoneGap havaitsee, että akun lataustaso on saavuttanut matalan lataustason
- **batterystatus:** laukeaa, kun PhoneGap havaitsee, että on tapahtunut muutos akun lataustasossa
- **menubutton:** laukeaa, kun käyttäjä painaa laitteen valikko-näppäintä
- **searchbutton:** laukeaa, kun käyttäjä painaa laitteen etsi-näppäintä
- **startcallbutton:** laukeaa, kun käyttäjä painaa laitteen soitonaloitus-näppäintä
- **endcallbutton:** laukeaa, kun käyttäjä painaa laitteen soitonlopetus-näppäintä
- **volumedownbutton:** laukeaa, kun käyttäjä painaa laitteen äänen vähentämisen näppäintä
- **volumedownupbutton:** laukeaa, kun käyttäjä painaa laitteen äänen nostamisen näppäintä

```
// Asetetaan kuuntelija PhoneGapin lataus tapahtumaan,
// joka laukeaa, kun PhoneGap on täysin ladattu.
// Kaikkien eri tapahtumien kuuntelijat asetetaan samaan tapaan.
document.addEventListener('deviceready', onDeviceReady, false);

// Näytetään käyttäjälle PhoneGap ilmoitus, kun PhoneGapin lataus on valmis.
function onDeviceReady() {
    navigator.notification.alert('PhoneGap on valmis.', null, 'Yeah.');
}
```

Kuva 17. Esimerkki Events-kirjaston käytöstä.

Tässä on linkki Events-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_events_events.md.html#Events

5.2.9 File

File-kirjasto sisältää tiedostojen käsittelyyn tarkoitetut toiminnot. Tiedostoja käsitellään laitteen käyttöjärjestelmän applikaatiolle varaamassa ns. hiekkalaatikossa, johon muut applikaatiot eivät pääse käsiksi. File-kirjastolla voidaan lukea, kirjoittaa, lähettää ja ladata palvelimelta tiedostoja, sillä voidaan myös kirjoittaa, joko pysyvään tai tilapäiseen tallennuspaikkaan.

Kuvassa 18 on esimerkki File-kirjaston käytöstä tiedoston latauksesta palvelimelle. Virheen tapahtuessa näytetään ilmoitus käyttäjälle.

```
// Ladataan tiedosto palvelimelle
function LataaTiedostoPalvelimelle(mediaTiedosto) {
    var tiedostoLataus = new FileTransfer(),
        path = mediaTiedosto.fullPath,
        name = mediaTiedosto.name;

    tiedostoLataus.upload(path,
            'http://www.osoite.com',
            TiedostonLatausOnnistui,
            TiedostonLatausEpaonnistui,
            { fileName: name });
}

// Lataus onnistui, näytetään tuloksista HTTP-tulos koodi PhoneGap ilmoituksena.
function TiedostonLatausOnnistui(result) {
    navigator.notification.alert('Lataus onnistui! \n' + result.responseCode, null, 'Hiiohoi!');
}

// Lataus epäonnistui, näytetään virhe käyttäjälle PhoneGap ilmoituksena.
function TiedostonLatausEpaonnistui(error) {
    var virheet = {};
    virheet[FileTransferError.FILE_NOT_FOUND_ERR] = 'Tiedostoa ei löydy';
    virheet[FileTransferError.INVALID_URL_ERR] = 'Väärä URI palveluun';
    virheet[FileTransferError.CONNECTION_ERR] = 'Yhteydessä vikaa';
    virheet[FileTransferError.ABORT_ERR] = 'Lataus keskeytyi';

    navigator.notification.alert('Lataus epäonnistui! \n' + virheet[error.code]);
}
```

Kuva 18. Esimerkki File-kirjaston käytöstä tiedoston latauksesta palvelimelle.

Tässä on linkki Files-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_file_file.md.html#File

5.2.10 Geolocation

Geolocation-kirjasto sisältää laitteen sijainnin paikantamisen mahdollistavat toiminnot. Riippuen laitteesta, sen sijainti voidaan joko paikantaa GPS:n tai lait-

teen yhteyksien; IP-osoite, RFID, langaton internetin, puhelinyhteyksien ja Bluetoothin perusteella.

Haku

Nykyinen paikka haetaan geolocation.getCurrentPosition-funktiolla, joka palauttaa Position-olion, joka sisältää sijainnin tiedot; leveysasteen, pituusasteen, korkeuden meren pintaan nähden, sijainnin tarkkuuden, korkeuden tarkkuuden, suunnan, nopeuden ja aikaleiman.

Kuvassa 19 on esimerkki Geolocation-kirjaston käytöstä etsittäessä nykyistä paikkatietoa. Virheen tapahtuessa paikkatietojen haussa näytetään käyttäjälle siitä ilmoitus.

Kuuntelu

Sijainnin kuuntelu hoidetaan geolocation.watchPosition-funktiolla, jolla voidaan seurata sijainnin vaihtelua tietyllä aikavälillä. Jokaisella kertaa, kun sijainti saadaan, kuuntelijalle annetaan Position-olio, joka sisältää sen hetkisen sijainnin tiedot.

Kuuntelun sulkeminen

Kuuntelu voidaan sulkea, kun näin halutaan, jotta ei vietäisi resursseja turhaan laitteelta, sillä sijainnin kuuntelu on hyvinkin resursseja vievää toimintaa, niin akulta kuin itse laitteen omilta resursseilta.

```
// Haetaan paikkatiedot
function HaeGeolocation() {
    var options = { enableHighAccuracy: true };
    navigator.geolocation.getCurrentPosition(onGeolocationSuccess, onGeolocationError, options);
}

// Mikäli paikkatietojen haku onnistuu, näytetään paikkatietojen tiedot.
function onGeolocationSuccess(position) {
    $('#divGeolocation').append('Latitude: ' + position.coords.latitude + '<br />'
            + 'Longitude: ' + position.coords.longitude + '<br />'
            + 'Altitude: ' + position.coords.altitude + '<br />'
            + 'Accuracy: ' + position.coords.accuracy + '<br />'
            + 'Altitude Accuracy: ' + position.coords.altitudeAccuracy + '<br />'
            + 'Heading: ' + position.coords.heading + '<br />'
            + 'Speed: ' + position.coords.speed + '<br />'
            + 'Aika: ' + position.timestamp)
}

// Mikäli tulee virhe paikkatietojen haussa, näytetään PhoneGap Ilmoitus.
function onGeolocationError(error) {
    navigator.notification.alert('Virh paikkatietojen haussa.', null, 'Oh noes!');
}
```

Kuva 19. Esimerkki Geolocation-kirjaston käytöstä.

Tässä on linkki Geolocation-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_geolocation_geolocation.md.html#Geolocation

5.2.11 Globalization

Globaalization-kirjaston kautta voidaan tehdä erinäisiä toimintoja, jotka riippuvat käyttäjän valitsemasta lokaatiosta laitteella sekä ajasta. Eri mailla on erilaiset numerojärjestelmät, joissakin käytetään desimaalina pilkkua ja joissakin pistettä. Päivämääriäkin näytetään eri tavalla eri maissa. Näiden ns. lokaalisten erojen vuoksi on kehitetty erilaisia kirjastoja, jotka auttavat muun muassa vaihtamaan päivämäärää ja lukujen näyttämistä käyttäjän valitseman paikkatiedon mukaan. Vaikka koodissa niitä käsitellään samalla lailla ilman, että niitä käännettäisiin lokaalista toiseen kesken laskujen, vaan laskut hoidetaan normaalisti ja lopputulos käännetään käyttäjän lokaaliversioon.

Globalization-kirjastolla voidaan tarkastaa käyttäjän laitteelle asettamia lokaalisaatio asetuksia, kääntää lokaalista toiseen numeroita, päivämääriä, valuuttoja, tarkkailla mikä on käyttäjän viikon ensimmäinen päivä, onko käyttäjän lokaalisaatiossa käytössä kesä/talviaika, kääntää numero merkkijonoksi ja toisinpäin.

Kuvassa 20 on esimerkki Globalization-kirjaston käytöstä, kun halutaan muuttaa numero paikalliseksi valuutaksi. Kuva sisältää myös esimerkin, kun halutaan muuttaa nykyinen päivä paikallisen muotoon.

```
// Muuntaa numeron lokaaliksi valuutaksi.
function NumeroValuuttaMerkkijonoksi(numero) {
    navigator.globalization.numberToString(numero,
            function (valuutta) { navigator.notification.alert('Valuuttana: ' + valuutta, null, 'Yeah!'); },
            function () { navigator.notification.alert('Virhe käännettäessä numeroa paikalliseksi valuutaksi.', null, 'Oh noes!'); },
            { type: 'currency' }
    );
}

// Näyttää käyttäjälle kuluvan päivämäärän.
function DateNow() {
    navigator.globalization.dateToString(new Date(),
            function (pvmString) { navigator.notification.alert('Päivämäärä: ' + pvmString, null, 'Yeah.'); },
            function (error) { navigator.notification.alert('Virhe käännettäessä päivämäärää paikalliseksi päivämääräksi.', null, 'Oh noes!')},
            { formatLength: 'short', selector: 'date and time' }
    );
}
```

Kuva 20. Esimerkki Globalization-kirjaston käytöstä.

Tässä on linkki Globalization-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_globalization_globalization.md.html#Globalization

5.2.12 InAppBrowser

InAppBrowser on internetselain, joka näytetään applikaatiossa, kun kutsutaan javascriptin window.open-funktiota, joka aukaisee uuden selainikkunan käyttöön haluttuun URIiin eli internetosoitteeseen. Tämä uusi luotu ikkuna voidaan sulkea ja sen tapahtumia voidaan kuunnella, jolloin voidaan tehdä jotain, kun tietty tapahtuma laukeaa. InAppBrowseriin voidaan lisätä kuuntelija, joka kuuntelee sivun latautumista, sivun latauksen loppua ja selaimen sulkemista.

Kuvassa 21 on esimerkki InAppBrowserin käytöstä. Esimerkissä asetetaan kuuntelija exit-tapahtumalle InAppBrowserille. Tapahtuman lauettua näytetään ilmoitus käyttäjälle.

```
// Odotetaan, että PhoneGap on latautunut.
document.addEventListener("deviceready", onDeviceReady, false);

// Aukaistaan uusi InAppBrowser selain ikkuna, joka ohjataan Googlen hakupalveluun
// ja lisätään sille kuuntelija, joka laukeaa, kun ikkuna suljetaan.
function onDeviceReady() {
    var ref = window.open('http://www.google.fi', '_blank', 'location=yes');
    ref.addEventListener("exit", function() { navigator.notification.alert('InAppBrowser suljettiin.', null, 'Sulki'); });
}
```

Kuva 21. Esimerkki InAppBrowserin käytöstä.

Tässä on linkki InAppBrowserin yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_inappbrowser_inappbrowser.md.html#InAppBrowser

5.2.13 Media

Media-kirjastolla hoidetaan äänitiedostoon kohdistuvat toiminnot, kuten toisto, toiston tauottaminen, toiston lopettaminen, äänitiedoston toiston sijainnin ja pituuden hakeminen ja äänen nauhoittaminen.

Kuvassa 22 on esimerkki Media-kirjaston käytöstä. Esimerkissä haetaan äänitiedosto palvelimelta, joka soitetaan. Soitto voidaan myös lopettaa esimerkin StopMediaFilePlaying-funktiolla. Virheen sattuessa näytetään käyttäjälle ilmoitus.

```
var mediaFile = null;

// Haetaan netistä mediatiedosto soitettavaksi ja soitetaan se.
function PlayMediaFileFromNet() {
    mediaFile = new Media("http://audio.ibeat.org/content/p1rj1s/p1rj1s_-_rockGuitar.mp3",
            onMediaSuccess,
            onMediaError);

    mediaFile.play();
}

// Mediafilen haku onnistui, kirjoitetaan konsoliin tieto siitä.
function onMediaSuccess() {
    console.log("MediaPlay Success Yeah!");
}

// Mediafilessä tapahtui virhe, näytetään käyttäjälle PhoneGap ilmoitus.
function onMediaError(error) {
    navigator.notification.alert('Tapahtui virhe mediatiedostossa.', null, 'Oh noes!');
}

// Lopetaan median toistaminen.
function StopMediaFilePlaying() {
    if (mediaFile) {
        mediaFile.stop();
    }
}
```

Kuva 22. Esimerkki Media-kirjaston käytöstä.

Tässä on linkki Media-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_media_media.md.html#Media

5.2.14 Notification

Notification-kirjastolla hoidetaan käyttäjälle tapahtumista tiedottaminen ja käyttäjän toimintojen vahvistaminen. Sillä voidaan ilmoittaa tapahtumista värinällä, piippauksella tai ilmoituksella. Kuvassa 23 on esimerkit kaikista näistä mahdollisuuksista ilmoittaa käyttäjälle tapahtumista.

Notification-kirjastoa ei saa sekoittaa älypuhelimissa olevaan palvelimelta tulevaan ilmoitukseen tai ilmoitukseen, kun sovellus on tausta-ajossa tai suljettuna. Notification-kirjastolla voidaan hoitaa vain käyttöliittymän ilmoituksia käyttäjälle.

```
// Näytetään ilmoitus käyttäjälle.
function NaytaIlmoitus(ilmoitus) {
    navigator.notification.alert(ilmoitus, null, 'Ilmoitus');
}

// Näytetään vahvistus käyttäjälle toiminnosta.
function NaytaVahvistus(vahvistus) {
    navigator.notification.confirm(vahvistus,
            function (buttonIndex) { console.log('ButtonIndex: ' + buttonIndex); },
            'Vahvistus',
            'OK,Peruuta'
    );
}

// Piipataan käyttäjälle kahdesti.
function Piippaa() {
    navigator.notification.beep(2);
}

// Väristä puhelinta 1 sekuntti.
function Varise() {
    navigator.notification.vibrate(1000);
}
```

Kuva 23. Esimerkki Notification-kirjaston käytöstä.

Tässä on linkki Notification-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_notification_notification.md.html#Notification

5.2.15 Splashscreen

Splashscreen-kirjastolla voidaan näyttää tai piilottaa sovelluksen latauskuva, joka on yleisemmin käytössä, kun sovellusta ensimmäistä kertaa käynnistetään.

Kuvassa 24 on esimerkki alkukuvan piilotuksesta, kun PhoneGap on täysin latautunut.

```
/*
*   SplashScreeni eli alkunäyttö kopioidaan res/drawable hakemistoon, josta PhoneGapin SpalshScreen sitä käyttää.
*   Kuvan koko voi olla joku seuraavista:
*
*   xlarge (xhdpi): vähintään 960 x 720
    large (hdpi): vähintään 640 x 480
    medium (mdpi): vähintään 470 x 320
    small (ldpi): vähintään 426 x 320

    Applikaation aloittavaan metodiin pitää lisätä
        super.setIntegerProperty("splashscreen", R.drawable.splash);
        super.loadUrl(Config.getStartUrl(), 10000);
    jotta alkunäytön käyttö onnistuu.
*/

// Odotetaan kunnes PhoneGap on valmis.
document.addEventListener("deviceready", onDeviceReady, false);

// Kun PhoneGap on valmis, piilotetaan alkunäyttö
function onDeviceReady() {
    navigator.splashscreen.hide();
}
```

Kuva 24. Esimerkki Splashscreen-kirjaston käytöstä.

Tässä on linkki Splashscreen-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_splashscreen_splashscreen.md.html#Splashscreen

5.2.16 Storage

Storage-kirjastolla hoidetaan tiedon varastointi tietokantaan ja tietokannasta tiedon hakeminen. Sen kautta voidaan myös hoitaa localStoragen käyttö, joka on pääasiassa asetuksia varten ja muuta pientä tallennusta vaativiin toimenpiteisiin tarkoitettu taulukko tyyppinen tallennusmuoto, johon tallennetaan tietoa merkkijonoina. Eli jos tallennetaan totuusarvoja tai lukuja, niin ne pitää konvertoida takaisin omaan tyyppiin, kun ne on haettu localStoragesta. Tietokantaan varastoidessa käytetään aina transaktioita eli joudutaan kirjoittamaan SQL lauseita, jotta saadaan tietokannasta jotain irti.

Haku

Kuvassa 25 on esimerkki miten voidaan hakea tietokannasta tietoa PhoneGapilla. Esimerkissä haetaan kaikki tieto taulusta Ostoslista, joka kyselyn onnistuessa näytetään käyttäjälle listana.

```
// Haetaan tietokannasta Ostoslista-taulusta
//
function queryDB(tx) {
    tx.executeSql('SELECT * FROM Ostoslista', [], querySuccess, errorCB);
}

//Kun Ostoslista-taulun haku on valmis, lisätään tavarat taulukkoon.
//
function querySuccess(tx, results) {
    var len = results.rows.length;
    console.log("Ostoslista table: " + len + " rows found.");
    $("#ostoslista").find("li").remove();
    for (var i=0; i<len; i++){
        console.log("Row = " + i + " ID = " + results.rows.item(i).id + " Data =  " + results.rows.item(i).data);
        $("#ostoslista").append('<li id="listItem" data-id="'+results.rows.item(i).id+'"><a href="#"><span>'+results.rows.item(i).data+'</span></a></li>');
    }
    $("#ostoslista").listview('refresh');
}
```

Kuva 25. Esimerkki Storagen käytöstä tietokannasta haettaessa.

Lisäys

Kuvassa 26 on esimerkki miten voidaan lisätä tietokantaan tietoa PhoneGapilla. Tieto lisätään PhoneGap Demo – tietokannan Ostoslista-tauluun.

```
// Lisätään tietokantaan Ostoslista-tauluun
//
function AddItem(tx, value) {
    tx.executeSql('INSERT INTO Ostoslista (data) VALUES (?)', [value]);
}

// Lisätään tietokantaan tavaraa
function addItemToDb(value) {
    var db = window.openDatabase("Database", "1.0", "PhoneGap Demo", 200000);
    db.transaction(function(tx) { AddItem(tx, value); }, errorCB, successCB);
}
```

Kuva 26. Esimerkki Storagen käytöstä tietokantaan lisättäessä tavaraa.

Päivitys

Kuvassa 27 on esimerkki miten voidaan päivittää tietoa tietokantaan PhoneGapilla. Esimerkissä päivitetään tietoa PhoneGap Demo –tietokannan Ostoslista-tauluun.

```
// Päivitetään Ostoslista taulun rivi
function UpdateItem(tx, id, value) {
    tx.executeSql('UPDATE Ostoslista SET data = ? WHERE id = ?', [value, id]);
}

// Päivitetään taulu tietokantaan
function UpdateItemInDb(id, value) {
    var db = window.openDatabase("Database", "1.0", "PhoneGap Demo", 20000);
    db.transaction(function(tx) { UpdateItem(tx, id, value); }, errorCB, successCB);
}
```

Kuva 27. Esimerkki Storagen käytöstä tietokantaan taulua päivittäessä.

Poisto

Kuvassa 28 on esimerkki miten voidaan poistaa tietoa tietokannasta PhoneGapilla. Esimerkissä poistetaan tietoa PhoneGap Demo –tietokannasta Ostoslista-taulusta.

```
// Poistetaan Ostoslista taulusta
function RemoveItem(tx, value) {
    tx.executeSql('DELETE FROM Ostoslista WHERE id = ?', [value]);
}

// Poistetaan tietokannasta tavaraa
function RemoveItemFromDb(value) {
    var db = window.openDatabase("Database", "1.0", "PhoneGap Demo", 200000);
    db.transaction(function(tx) { RemoveItem(tx, value); }, errorCB, successCB);
}
```

Kuva 28. Esimerkki Storagen käytöstä tietokantaan taulusta poistattessa.

Käsittely

Kuvassa 29 on esimerkki, miten voidaan hoitaa virheiden käsittely ja transaktioiden onnistuminen PhoneGapissa. Esimerkissä kirjoitetaan konsoliin tieto epäonnistumisesta, joka virheiden etsinnän aikana huomataan konsolista, kun virhe tapahtuu. Transaktion onnistuessa yleensä voidaan näyttää käyttäjälle ilmoitus tai kuten esimerkissä näytetään lisätyt tai muuttuneet tiedot käyttäjälle.

```
// Jos tulee virhe tietokannan käytössä, kirjoitetaan konsoliin virhe.
//
function errorCB(err) {
    console.log("Error processing SQL: "+err.code);
}

// Kun kannan taulujen toiminnot ovat valmiit haetaan tauluista uusimmat tiedot
//
function successCB() {
    var db = window.openDatabase("Database", "1.0", "PhoneGap Demo", 200000);
    db.transaction(queryDB, errorCB);
}
```

Kuva 29. Esimerkki onnistuneen ja epäonnistuneen transaktion käsittelystä Storagen kanssa.

Tässä on linkki Storage-kirjaston yksityiskohtaiseen dokumentaatioon:

http://docs.phonegap.com/en/2.4.0/cordova_storage_storage.md.html#Storage

6 PHONEGAP TYÖVÄLINEOHJELMISTOISSA

6.1 Tiedon säilytys

Läpikäydessäni PhoneGapia huomasin, että tiedon säilytyksessä ei ollut paljon eroja natiivin ohjelmistokehityksen ja PhoneGapin välillä. Molemmissa voidaan käyttää tietokantaa tiedon säilytykseen, jolloin tieto voidaan järjestää nopeammin järkeviin kokonaisuuksiin ja näin helpottaa kehitystyötä. Tiedon käsittely tietokannan kautta on kehittäjästä riippuen molemmilla natiivisti sekä PhoneGapilla yhtä helppoa, tosin PhoneGapilla pitää myös osata SQL-kysely kieli, jotta pystyy käyttämään tietokantaa.

PhoneGapin tarjoama SQL-tietokanta on itse asiassa HTML5-selaimissa oleva webSQL-tietokanta, jolloin taas liikutaan selaimen ehdoilla, mitä voidaan kehittää. Selaimesta riippuen tietokannalla on rajattu tallennuskapasiteetti, yleensä 5MT eli taas pitää pitää mielessä mobiilikehityksen ainainen ongelma eli resurssien koko. Pitää olla tarkkana siitä, mitä tallentaa ja kuinka paljon, jotta älypuhelimen käytettävyys ohjelman sisällä ja älypuhelimen oma käytettävyys ei häiriintyisi liikaa resursseja syövän ohjelman takia.

PhoneGapin localStroge on itse asiassa HTML5:n tukemista selaimista tuttu localStorage, joten selaimesta riippuen sen käyttö on aika rajoittunut. Joillakin selaimilla sen tallennuskapasiteetti on vain 2,5 MT, jolloin sinne ei paljon tavaraa saa tallennettua ja koska sinne voidaan tallentaa tietoa avainpareina merkkijonoina, localStoragea tulisi käyttää mieluiten vain asetusten tallennukseen.

Tietoa voidaan tallentaa myös suoraan tiedostoihin PhoneGapin File-kirjaston avulla. File-kirjaston kanssa ei enää liikuta selaimen rajoitusten mukaan, sillä se tallentaa tiedostonsa laitteen sovellukselle varattuun hiekkalaatikkoon. Mutta koska työvälineohjelmistoissa yleensä halutaan käyttää hyödyksi palvelimella olevaa tietoa, niin tiedon väliaikainen tai synkronointia tarvitseva tieto kannattaa tallentaa mieluiten tietokantaan. Tällöin palvelimella oleva tietokanta ja älypuhelimessa oleva tietokanta voivat olla rakenteeltaan samanlaisia ja näin helpote-

taan tiedon synkronointia ja käsittelyä, koska sitä käsitellään eri tekniikoista huolimatta samaan tapaan.

6.2 Synkronointi

PhoneGapin kirjastoissa on huomattavissa, että PhoneGap on todellakin vain rajapintana laitteeseen päin. Jos haluaa synkronoida tietoja PhoneGapin antaman tiedon tallennuksen ja palvelimen välillä, käytetäänkin javascriptin omia tekniikoita. Näistä tämän hetken parhaimmat ovat Ajax ja JSON.

6.2.1 Ajax

Ajax on yksinkertaisuudellaan ja tehokkuudellaan paras keino pitää yllä eriävää tietoa lokaalin sovelluksen ja palvelimen välillä, sillä ajax tekee verkkokutsut taustalla eli se lähettää palveluun vain tarvittavan tiedon eikä tee kokonaista sivukutsua, jolloin selain päivittäisi koko sivun. Parhaimmillaan ajaxilla voidaan lähettää vain funktion nimi, parametrit ja muut palvelinkutsun tarpeelliset tiedot, pienentäen näin verkkoliikennettä, joka on hyvin tärkeää mobiililaitteiden maailmassa resurssien takia. Eritoten akun takia, sillä internetyhteys vie hyvin paljon virtaa ja nykypäivänkään mobiililaitteiden verkkoyhteydet eivät ole kovin nopeita, joten käyttömukavuuden kannaltakin kannattaa käyttää ajaxia palvelimen ja sovelluksen välillä. [24]

Ajaxin kanssa voi myös lähettää kirjautumisotsikkotiedot, jotta ei unohdettaisi palveluissa tietoturvaa, sillä eihän kukaan halua, että kirjautumista vaativaan järjestelmään pääsisi kuka tahansa käsiksi.

Kuvassa 30. on esimerkki ajax kutsusta. Esimerkissä kutsulle annetaan asetukset, joiden mukaan se toimii. Osoite, josta tietoa halutaan kysellä, kyselyn tyyppi, kyselyn parametrit data-attribuutissa ja parametrien tyyppi. Ennen kyselyä asetetaan kyselylle tunnistautumisotsikko, jolla palvelin voi todentaa kyselyn olevan sallittu. Virheen sattuessa tai onnistumisessa näytetään ilmoitus käyttäjälle tapahtuneesta.

```
$.ajax( {
            url : 'http://www.something.com/wcf/menu',
            type: 'POST',
            data : {menuitems: JSON.stringify(jsonMenuitems) },
            dataType : 'json',
            beforeSend : function(xhr) {
                        var cryptedHMACStr = "2A43CF3F5FB6C";
                        xhr.setRequestHeader("Authorization", "Basic " +
                        cryptedHMACStr);
            },
            error : function(xhr, ajaxOptions, thrownError) {
                        alert('Ajax kutsu ei onnistunut palveluun.')
            },
            success : function(model) {
                        alert('Ajax kutsu onnistui palveluun.');
            }
});
```

Kuva 30. Esimerkki ajax-kutsusta.

Tähän Ajax-kutsuun on käytetty jQueryn ajaxkutsua tehden siitä yksinkertaisemman kuin javascriptin oma ajaxtekniikka. Ajaxkutsulle voi antaa erinäisiä attribuutteja, joiden mukaan selain kommunikoi palvelimen kanssa. Tässä esimerkissä on yleisimmät attribuutit ajaxkutussa. URI eli osoite, johon halutaan tehdä kysely, type kyselyn tyyppi, data kyselyn tieto, joka lähetetään palvelimelle käsiteltäväksi ja datatype tiedon tyyppi, joka lähetetään palvelimelle.

JQueryn ajaxkutsuun voi liittää myös erilaisiin tapahtumiin kuuntelijoita. BeforeSend tapahtuma laukaistaan ennen kuin pyyntö lähtee palvelimelle, esimerkissä lisätään kirjautumisotsikko pyyntöön mukaan, jolla voidaan palvelimen päässä todeta, että lähettäjällä on oikeus kyseiseen pyyntöön. Error tapahtuma laukaistaan, kun ajaxpyyntö ei onnistu syystä tai toisesta, esimerkissä ei käsitellä palautetun virheen tietoja vaan näytetään käyttäjälle ilmoitus epäonnistuneesta kutsusta. Success tapahtuma laukaistaan, kun ajaxpyyntö onnistui ja palvelimelta saadaan vastaus takaisin, jolloin voidaan käsitellä palvelimelta tullut vastaus, esimerkissä annetaan vain onnistuneen pyynnön tieto käyttäjälle ilmoituksena.

6.2.2 JSON

Ajaxin kanssa kannattaa käyttää XML merkkikielen sijaan JSONia, koska mobiililaitteiden rajallisten verkkoyhteysnopeuksien takia verkkoyhteyttä ei sovi rasittaa. Se kuluttaa akkua paljon ja tekee käyttäjän käyttökokemuksesta huonompaa, kun toimenpiteet palvelimen ja asiakasohjelman välillä ovat hitaita ja kömpelöitä.

JSON on tapa esittää olio merkkijonona. JSONissa erona perinteiseen XML-merkkikieleen on se, että olion esittäminen merkkijonona tehdään yksinkertaisemmin ja tilaa säästäen. XML sisältää kaikki tiedot jokaisesta olion attribuutista, jolloin se vie paljon tilaa. JSON esittää vain olion attribuutin nimen ja sen tiedon, jolloin olion kuvaus merkkijonona vie vähemmän tilaa. [25]

Kuvassa 31. on esimerkki json merkkijonosta, joka sisältää menuitem tyyppisen olion tiedot id ja nimi.

```
{
            "menuitem": {
                        "id": 1,
                        "nimi": "maito"
            }
}
```

Kuva 31. Esimerkki json-merkkijonosta.

7 YHTEENVETO

Liikemaailmassa on paljon kilpailua, sillä ihmiset ovat erilaisia ja haluavat erilaisia tuotteita. Jotkut tekevät asiat paremmin kuin toiset, toiset tuovat markkinoille halvempaa tuotetta, vaikka olisikin muuten samankaltainen toisen tuotteen kanssa. Kun on löytänyt tavan tehdä jonkin asian nopeammin ja halvemmin, niin sillä luodaan se kilpaileva tuote.

Laitemaailmassa nämä eroavaisuudet ovat vielä isompia, kun ajatellaan ohjelmointia. Eri laitteilla on eri käyttöjärjestelmät, eri toimintaympäristöt ajatellen ohjelmistojen julkaisua ja tuottamista, eri ohjelmointikielet, joilla on erilaiset mahdollisuudet tehdä ohjelmia. Kaikki nämä erilaisuudet heijastuvat saman ohjelman tekoon eri laitteille ja koska elämme kilpailuyhteiskunnassa, jossa ajatellaan ensin rahaa ja heti tulevaa säästöä, jolloin haemme ratkaisua laitteet yhdistävästä tekniikasta.

Tutustuin PhoneGapiin tämän tienoilta, joka antaa mahdollisuuden tehdä HTML5, CSS ja Javascriptin avulla ohjelmia älypuhelimiin. Tänä aikana, kun tutustuin PhoneGapiin minulle tuli selväksi, se ei ole mikään ihme tekniikka, joka pelastaisi yritykset tässä erilaisuuden maailmassa tarjoamalla täydellisen rajapinnan eri laitteiden välillä. Sillä PhoneGap käyttää hyväksi omassa tekniikassaan älypuhelimen selainta ja näin siinä ilmenee selaimen tuomat heikkoudet, ajatellen mitä voidaan tuoda käyttäjälle ilman, että hyödynnetään käyttöjärjestelmän valmistajien tuomia mahdollisuuksia.

PhoneGapissa voidaan tehdä kaikki se mitä selaimelle tehdyssä mobiilisivustolla voidaan tuoda käyttäjälle, sillä selain on vain käyttöliittymärajapinta ja näin selaimen ja laitteen valmistaja määrää sen missä suhteessa selaimella voidaan käskyttää laitetta. On ilmeistä, että tausta-ajon aikana tapahtuvia toimintoja ei voi tehdä selaimen päälle tai palvelimelta tulevien ilmoitusten käsittelyä, vaan ne pitää tehdä natiivisti.

Kävin PhoneGapia lävitse kahden ohjelman kautta, molemmat osoittautuivat hyviksi tavoiksi käydä uusi tekniikka lävitse. Ostoslista-ohjelmassa sain käsityk-

sen siitä, että PhoneGapissa ei ole kaikkea mitä siitä voisi heti ajatella, vaan esim. synkronointiin pitää käyttää toisia tekniikoita hoitamaan tämä alue. PhoneGapin kokonaisuuden läpikäymiseen tekemäni ohjelman aikana tämä asia taas korostui ja selvensi vielä enemmän, missä oikeastaan PhoneGapissa on kyse.

Mikäli harkitsee PhoneGapin käyttöä oman sovelluksen perustana, pitää käydä sovelluksen tarpeet lävitse yksitellen ja verrata tarpeita PhoneGapin mahdollisuuksiin. PhoneGapin hyviin puoliin kuuluu nopea kehitys eri älypuhelin alustoille, mutta huonoihin puoliin kuuluu selainpohjainen kehitystyö. Selainpohjainen kehitystyö sulkee pois älypuhelinalustojen tarjoamia vahvuuksia, kuten intuitiivisen käyttökokemuksen sovellusten välillä, tausta-ajon aikana tapahtuvat toimenpiteet, kuten ilmoitusten lähettämisen palvelimelta sovellukselle ja selain hidastaa ohjelman toimintaa, joka pienentää käyttökokemusta entisestään. Tietoturvaa ei tässä dokumentissa käsitelty tai käyttöliittymä puolta, vaikka ne kuuluvat oleellisesti PhoneGapin käyttöön sovelluksissa, koska ne eivät tähän opinnäytetyöhön mahtuneet. Nekin pitää muistaa, kun tehdään päätöstä mitä tekniikkaa kannattaisi käyttää älypuhelinsovelluksen toteuttamiseen. Näiden hyvien ja huonojen puolien takia suosittelisin PhoneGapia käytettäväksi yksinkertaisiin ja ei niin resursseja vaativiin ohjelmiin, sillä monimutkaisimmissa ohjelmissa, jotka tarvitsevat enemmän resursseja tarvitaan käyttömukavuuden kannalta tehokkuutta ja tietoturvan kannalta ei niin läpinäkyvää toiminnallisuutta, jota ei selaimella voida tehdä kuin tiettyyn pisteeseen asti.

LÄHTEET

[1] jQuery Mobile 2013 mobiili ohjelmistokehys käyttöliittymälle, viitattu 27.3.2013

http://jquerymobile.com/

[2] jQuery 2013, javascript kirjasto, viitattu 27.3.2013

http://jquery.com/

[3] iOs 2011, Wikipedia, viitattu 2.12.2011.

http://en.wikipedia.org/wiki/IOS

[4] Android 2011, Wikipedia, viitattu 2.12.2011

http://en.wikipedia.org/wiki/Android_(operating_system)

[5] Windows Phone 7 2011, Wikipedia, viitattu 2.12.2011.

http://en.wikipedia.org/wiki/Windows_phone_7

[6] Työvälineohjelmat 2011, viitattu 2.12.2011

http://me-tietotekniikka.wikispaces.com/johanna_3.tyovalineohjelmat

[7] Android Data Storage 2011, viitattu 2.12.2011.

http://developer.android.com/guide/topics/data/data-storage.html#netw

[8] Window Phone 7 2011, Developer Guide, viitattu 13.11.2011.

http://msdn.microsoft.com/en-us/library/gg490765.aspx

[9] Mobile Application Development 2011, viitattu 2.12.2011.

http://en.wikipedia.org/wiki/Mobile_application_development

[10] Developer Tools, 2011, viitattu 30.11.2011.

http://developer.apple.com/technologies/tools/

[11] iCloud for Developer 2013, viitattu 6.6.2013.

http://developer.apple.com/icloud/index.php

[12] iOs Developer Library 2011, viitattu 12.11.2011

http://developer.apple.com/library/ios/navigation/

[13] Pilone, Tracey; Pilone, Dan 2009. Head First iPhone Development. California: O'Reilly Media.

[14] JSON and iOs 5 2011, viitattu 1.12.2011.

http://www.raywenderlich.com/5492/working-with-json-in-ios-5

[15] Java Programming Language 2011, Wikipedia, viitattu 2.12.2011.

http://en.wikipedia.org/wiki/Java_(programming_language)

[16] Microsoft Silverlight 2011, Wikipedia, viitattu 30.11.2011.

http://en.wikipedia.org/wiki/Microsoft_Silverlight

[17] Microsoft XNA, Wikipedia, viitattu 30.11.2011.

http://en.wikipedia.org/wiki/Microsoft_XNA

[18] Artikkeli lokaalisesta tietokannasta Windows Phone 7:ssä 2011, viitattu 30.11.2011.

http://windowsphonegeek.com/articles/Windows-Phone-Mango-Local-Database-mapping-and-database-operations

[19] Windows Azure Platform 2011, viitattu 30.11.2011.

http://www.microsoft.com/windowsazure/

[20] Web Service Security for Windows Phone 2011, viitattu 30.11.2011.

http://msdn.microsoft.com/en-us/library/gg521147(v=VS.92).aspx

[21] Window Azure Toolkit for Windows Phone 2013, viitattu 6.6.2013.

http://watwp.codeplex.com/

[22] Phonegap Project 2013, viitattu 13.3.2013

http://phonegap.com/about/

[23] PhoneGap kääntäjä pilvipalveluna 2013, viitattu 25.3.2013

https://build.phonegap.com/

[24] Ajax Asynkronoitu javascript xml –tekniikka 2013, viitattu 13.4.2013

http://fi.wikipedia.org/wiki/Ajax_(ohjelmointi)

[25] JSON merkkikieli olioiden esittämiseen merkkijonona 2013, viitattu 13.4.2013.

http://en.wikipedia.org/wiki/JSON